MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

ChessEngine – Jucător Virtual de Șah

LUCRARE DE LICENŢĂ

Absolvent: **David Balazs**

Coordonator ştiinţific: **Ș. l. ing. Cosmina IVAN**

2016

MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

DECAN,
Prof. dr. ing. Liviu MICLEA

DIRECTOR DEPARTAMENT,
Prof. dr. ing. Rodica POTOLEA

Absolvent: **David Balazs**

ChessEngine – Jucător Virtual de Șah

1. **Enunţul temei:** *Proiectul isi propune realizarea unui jucator virtual menit sa ofere suport de antrenament jucatorilor de șah, precum si mijloc de divertisment pentru jucatorii ocazionali.*

2. **Conţinutul lucrării:** Cuprins, Introducere, Obiectivele Proiectului, Studiu Bibliografic, Analiza si Fundamentare Teoretica, Proiectare si Implementare, Testare, Validare si Evaluare, Manual de Instalare si Utilizare, Concluzii, Bibiliografie

3. **Locul documentării**: *Exemplu*: Universitatea Tehnică din Cluj-Napoca, Departamentul Calculatoare

4. **Consultanţi**:

5. **Data emiterii temei:** 1 noiembrie 2015

6. **Data predării:** 30 Iunie 2016

Absolvent: Balazs David

Coordonator ştiinţific: Ș. l. ing. Cosmina IVAN

MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

Declaraţie pe proprie răspundere privind autenticitatea lucrării de licenţă

Subsemnatul **Balazs David**, legitimat cu cartea de identitate seria BV nr. 769844 CNP 1930729080011, autorul lucrării ChessEngine – Jucator Virtual de Şah elaborată în vederea susţinerii examenului de finalizare a studiilor de licență la Facultatea de Automatică şi Calculatoare, Specializarea Calculatoare din cadrul Universităţii Tehnice din Cluj-Napoca, sesiunea Iulie a anului universitar 2015-2016, declar pe proprie răspundere, că această lucrare este rezultatul propriei activităţi intelectuale, pe baza cercetărilor mele şi pe baza informaţiilor obţinute din surse care au fost citate, în textul lucrării, şi în bibliografie.

Declar, că această lucrare nu conţine porţiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislaţiei române şi a convenţiilor internaţionale privind drepturile de autor.

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

In cazul constatării ulterioare a unor declaraţii false, voi suporta sancţiunile administrative, respectiv, *anularea examenului de licenţă*.

Data _____

Nume, Prenume _____

Semnătura

Cuprins

Capitolul 1. Introducere – Contextul proiectului

În capitolul ce urmează se va face o introducere a aplicației ChessEngine, prin prezentarea contextului în care a apărut ideea dezvoltării acestei aplicații, precum și motivația. De asemenea, va fi prezentat și conținutul lucrării pe scurt.

1.1. Contextul proiectului

Odată cu evoluția tehnologiei au apărut mecanisme software menite să ușureze muncă de zi cu zi a oamenilor. Astfel, au apărut editoare performanțe de documente, utilitare pentru organizarea taskurilor în companii, platforme de comerț electronic ce oferă posibilitatea plasării comenzilor și procesării plăților direct prin intermediul internetului. Pe de altă parte, au fost dezvoltate și aplicații software menite să creeze din sistemele electronice niște mijloace de divertisment. Astfel au apărut rețele sociale, unde utilizatorii pot poștă diferite mesaje, poze sau filmări, pot comunică între ei, au apărut aplicații online ce permit vizualizări de filme. Un rol important în gamă de aplicații de divertisment îl au jocurile, acestea facundu-și apariția încă de la începuturile sistemelor electronice și evoluând odată cu acestea. Jocurile din zilele noastre folosesc concepte din inteligență artificială și modele din lumea reală pentru a aduce o experiență cât mai reală jucătorilor. Printre aceste jocuri, se număra și șahul. Încă de la începuturile sistemelor electronice, programarea unui jucător virtual de șah a prezentat o provocare pentru oamenii de știință. De-a lungul timpului au apărut mecanisme capabile să joace șah, acestea ajungând până la nivelul jucătorilor de top.

1.2. Motivatia

Ținând cont de faptul că șahul este un joc ce implică logică, stările sale sunt bine determinate, iar pentru fiecare stare există un set definit de mutări, implementarea unui jucător virtual de șah a reprezentat încă de la începutul sistemelor electronice o provocare pentru oamenii de știință. De-a lungul timpului au fost descoperite tehnici de căutare în spațiul de mutări, metode de îmbunătățire a deciziilor pe care jucătorul le ia în timpul meciului în alegerea unei mutări, iar în ultimul timp, jocul de șah reprezintă o arie de interes și pentru specialiștii în Machine Learning.

Pentru aplicația de față, ChessEngine, motivația a fost de a dezvoltă o platformă ce vine în ajutorul jucătorilor de șah, oferindu-le suport de antrenament prin expunerea unui astfel de jucător de șah, a unui set de probleme și strategii de șah. Aplicația va putea fi utilizată atât de jucători profesioniști de șah, cât și de jucători de ocazie, în scopul de divertisment.

1.3. Continutul lucrarii

În continuare vor fi prezentate capitolele, după cum urmareaza: se vor prezența obiectivele proiectului, ce stau în strânsă legătură cu motiatia acestuia, fiind țelurile ce se doresc a fi atinse. Următorul capitol va prezența studiul bibliografic ce să făcut înaintea dezvoltării aplicației, alternativele găsite, algoritmii de căutare și metodele de generare a următoarei mutări într-un joc de șah. În continuare vor fi prezentate aspecte de analiză și

fundamentare teoretică, aici fiind prezentate tehnologiile folosite, împreună cu posibilele alternative și argumentele pentru care s-a optat pentru o anumită tehnologie. Tot în acest capitol sunt prezentate cerințele funcționale și nonfunctionale, precum și actorii sistemului și cazurile de utilizare. În capitolul de "Proiectare de detaliu și implementare" este descrisă modalitatea în care a fost implementată aplicația, având 3 secțiuni importante: implementarea aplicației web, implementarea jucătorului virtual de șah și implementarea interfeței grafice. Tot în acest capitol este prezentată și metodologia folosită pentru dezvoltarea aplicației. În continuare se prezintă aspecte de testare a funcționalităților și țeste de performanță. Între ultimele capitole se află un manual de instalare și utilizare a aplicației și concluziile proiectului. În această ultima secțiune sunt prezentate obiectivele ce au fost realizate și posibilități de dezvoltare ulterioare.

Capitolul 2. Obiectivele Proiectului

În acest capitol vor fi prezentate obiectivele propuse spre a fi realizate. Sistemul are scopul de a oferi suport de antrenament jucătorilor de șah, precum și mijloc de divertisment pentru jucătorii neexperimentați.

2.1. Obiectivul principal

Obiectivul principal al acestui proiect reprezintă proiectarea, definirea și construirea unui sistem care să ofere suport pentru antrenamentul jucătorilor de șah, obiectiv care se poate atinge prin realizarea obiectivelor secundare care urmează a fi prezentate.

2.2. Obiective secundare

- **Obiective generale propuse**

Primul dintre obiectivele secundare pe care sistemul dorește să le atingă este ușurință în utilizare, obiectiv care se concretizează prin simplitatea interfeței grafice și numărul minim de pași pe care utilizatorul trebuie să îi parcurgă pentru a realiză o acțiune. De asemenea, pe paginile aplicației există un meniu special în care sunt puse legături către funcționalitățile principale ale aplicației.

Un alt obiectiv este proiectarea aplicației astfel încât această să fie permisibila la dezvoltări ulterioare, cum ar fi adăugarea de funcționalități noi, extragerea jucătorului virtual într-un serviciu Web separat sau schimbarea funcționalității existențe: înlocuirea bazei de date folosite sau a interfeței grafice. Pentru a îndeplini acest obiectiv trebuie că aplicația să fie modularizata iar componentele să aibă cât mai puține dependințe cu putință, folosind design pattern-uri existențe.

Un obiectiv legat de interfață grafica este acela că se dorește un design plăcut, care să pună conținutul în evidența. Acest obiectiv poate fi realizat prin utilizarea unui software dedicat desenării interfețelor grafice, cum ar fi Adobe Photoshop.

- **Jucarea unui meci de șah**

Posibilitatea jucătorului de a jucă un meci de șah împotrivă sistemului la un anumit nivel de dificultate, cu timp de răspuns ce se încadrează într-un anumit timp limită specificat. Acest obiectiv poate fi realizat prin implementarea unui jucător virtual folosind un algoritm eficient care pornind de la o poziție data pe tablă de șah generează un anumit număr de mutări în avans și o alege pe cea mai bună dintre ele. Mai multe nivele de dificultate pot fi obținute prin parametrizarea numărul de mutări pe care sistemul le va genera în avans. Spre exemplu, pentru nivelul de dificultate începător, sistemul va genera două mutări în avans în timpul procesării, iar pentru nivelul avansat, sistemul va genera zece mutări în avans. Timpul de răspuns poate fi influientat de către funcția de evaluare a unei poziții. Cu cât sunt mai multe criterii pentru evaluarea unei poziții, cu atât sistemul va fi mai lent, iar cu cât criteriile sunt organizate mai eficient, cu atât sistemul va fi mai rapid în găsirea unei mutări.

- **Vizualizarea de strategii de șah**

Funcționalitatea vizualizării de strategii de joc este un obiectiv ce vine în sprijinul jucătorilor ce doresc să își extindă cunostiintele în domeniul jocului de șah. Astfel se dorește implementarea unei funcționalități prin care utilizatorul are la dispoziție o lista de strategii, poate selectă una dintre ele și poate urmări succesiunea de mutări pe o tablă de șah.

- **Rezolvarea de probleme de șah**

Funcționalitatea ce include rezolvarea problemelor de șah este un alt obiectiv ce vine în sprijinul jucătorilor ce doresc să își extindă cunostiintele în domeniul jocului de șah. Se dorește expunerea funcționalității sub formă unei liste cu probleme de șah dintre care utilizatorul poate selectă una spre a o rezolvă. Problemă de șah va fi afișată sub formă unei table de șah pe care este setată poziția de șah corespunzătoare problemei și o cerința. Utilizatorul va face mutările necesare pe tablă de șah și va primi un rezultat pozitiv în cazul găsirii soluției sau un rezolvat negativ în cazul în care nu a ajuns la soluție.

Capitolul 3. Studiu Bibliografic

În continuare vor fi prezentate conceptele studiate cu scopul dezvoltarii aplicatiei pentru șah, ChessEngine.

3.1. Programarea jucătorului de șah

În acest capitol vor fi prezentate aspecte legate de jocul de șah, precum și posibilități de implementare a unui sistem ce are capabilitatea de a jucă șah și provocările aduse de implementări.

3.1.1. Jocul de șah

Șahul este un joc ce implică logică, imaginație și viziune din partea jucătorilor. La fiecare pas, jocul se află într-o stare bine definită: poziția pe tablă de șah este deterministă, iar unul dintre jucători, fie albul sau negrul trebuie să facă o mutare. Cazul special este acela în care meciul s-a terminat, fie datorită faptului că unul dintre jucători a câștigat, fie că meciul s-a terminat cu remiza (egalitate) fie unul dintre jucători a abandonat meciul.

O proprietate importantă a șahului este faptul că este un joc de tip „zero-sum”, deoarece câștigul unui jucător implică pierderea pentru celălalt jucător, prin urmare, interesele celor doi oponenți sunt diametral opuse.

O altă proprietate importantă a șahului este aceea că, pentru orice poziție pe tablă de șah, există un set finit de mutări posibile pentru jucătorul ce trebuie să facă o mutare.

Din punct de vedere computațional, jocul de șah este o sarcină complexă. Conform Wikipedia, numărul de jocuri de șah distincte este aproximativ 10^44. La oră actuală nu există sistem de calcul capabil să analizeze toate aceste jocuri de șah și să ofere un pronostic atunci când ambele părți joacă cele mai bune mutări [4].

3.1.2. Posibilități de implementare jucător virtual de șah

Jucătorul virtual de șah este un sistem capabil să joace un meci de șah cu un oponent. Că și parametrii de intrare, jucătorul virtual are nevoie de poziția curentă de pe table de șah și culoarea ce trebuie să mute (alb sau negru). Că și parametrii de ieșire, sistemul returnează mutarea următoare.

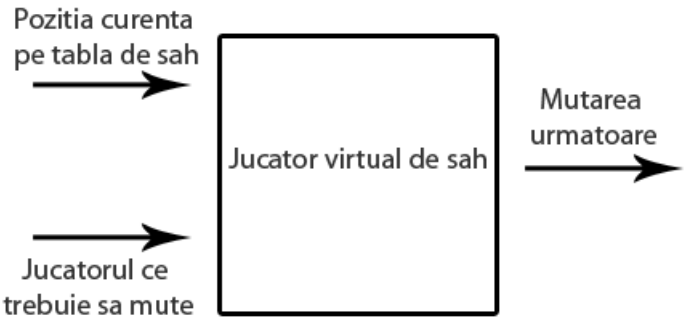

Figura 3.1 Arhitectura generală jucător virtual de șah

Pentru implementarea unui astfel de sistem există mai multe abordări, dintre care cele mai importante sunt calcul prin forță brută, machine learning și deep learning.

3.1.2.1. Găsirea urmatoarei mutări prin forță brută (algoritmul Minimax)

Această abordare presupune generarea următoarelor N mutări în avans și organizarea lor sub formă de arbore pentru o poziție data astfel: se generează toate mutările posibile pentru alb, apoi pentru fiecare mutare a albului, se generează toate mutările posibile ale negrului. Acești pași se repetă până se atinge numărul N de mutări. Toate aceste mutări sunt reprezentate sub formă de arbore, în care un nod reprezintă o poziție de șah, iar o legătură ce unește două noduri este o mutare. La final, toate nodurile frunză (pozițiile de șah) sunt evaluate iar poziția ce primește cel mai mare scor este aleasă, sistemul alegând succesiunea de mutări ce duce la această poziție [3].

Algoritmul Minimax

Un algoritm folosit pentrtu această abordare este Mini-Max. Acest algoritm are la bază ideea că jucătorul alb încearcă să își maximizeze șansele de câștig, la fel și jucătorul negru încearcă să își maximizeze șansele de câștig, ceea ce într-un jocde tip „zero-sum” cum este și șahul, este echivalent cu propoziția că jucătorul negru încearcă să minimizeze șansele jucătorului alb de a câștiga.

Algoritmul Mini-max presupune generarea arborelui cu mutări posibile și alegerea acelei secvențe ce maximizează șansele de câștig ale jucătorului pentru care se generează. În cadrul arborelui, jucătorul alb este denumit Max iar jucătorul negru este denumit Min, deoarece jucătorul alb (Max) dorește să maximizeze șansele de câștig ale albului, iar jucătorul negru (Min) dorește să minimizeze șansele de câștig ale albului [23].

Deoarece arborele poate să devină foarte mare după doar câteva mutări, de aceea se impune o anumită adâncime până la care se va dezvoltă arborele, această adâncime fiind defapt numărul de mutări generate în avans de către sistem. Importantă adâncimii de căutare vine din faptul că această este un factor important în performanță sistemului.

Statistic, având o poziție de șah data, există un număr de 20 de mutări posibile pentru jucătorul ce trebuie să facă o mutare. Dacă adâncimea de căutare este 2, prin urmare sistemul va genera 2 mutari in avans, atunci se vor genera aproximativ 20^2 mutări: pentru poziția inițială sunt 20 de mutări posibile, iar pentru fiecare poziție nouă, obținută prin aplicarea uneia dintre aceste 20 de mutări posibile, există alte 20 de mutări posibile.

Funcția de evaluare

Pentru fiecare frunză a arborelui (poziție pe tablă de șah) se aplică o funcție de evaluare al cărei rol este să estimeze valoarea acestei poziții din perspectivă jucătorului alb, prin urmare: dacă jucătorul alb este în avantaj față de cel negru, funcția va returna o valoare pozitivă pentru această poziție pe tablă de șah și invers, dacă negrul este în avantaj, funcția va returna o valoare negativă.

Sistemul folosește această funcție de evaluare pentru a compară pozițiile de șah între ele și pentru alegerea unei strategii corespunzătoare de joc.

Funcția de evaluare pentru o poziție de șah este reprezentată prin evaluarea unui set de criterii cum ar fi avantajul material, poziția pieselor de șah pe tablă, sau siguranță regelui [10].

În calculul avantajului material, se ia în considerare diferență dintre numărul pieselor din fiecare tip: numărul de pioni ai albului – numărul de pioni ai negrului, numărul de turnuri ai albului – numărul de turnuri ai negrului. De menționat este faptul că fiecare tip de piesă are o importantă diferită pe tablă de șah. Spre exemplu, un nebun este mai important decât un pion, dar mai puțin important decât o regină. Pentru a simulă acest efect, se folosesc ponderi asociate fiecărui tip de piesă, raportat la pion. Spre exemplu, un nebun poate fi considerat că valorând cât echivalentul a trei pioni. Că și observație, regele nu poate fi capturat, fiecare jucător având exact un rege, prin urmare, această piesă nu va fi luată în considerare pentru calculul avantajului material [22].

Lista de criterii ce va intră în componența funcției de evaluare va influiență în mod direct calitatea mutărilor sistemului, doarece o evaluare cât mai apropiată de realitate a unei poziții va duce la mutări mai bune.

Pentru evaluarea poziției pieselor de șah pe tablă, sunt multe aspecte ce trebuie luate în considerare. Piesele „ce aluneca" pe tablă (regină, turnul și nebunul) se consideră că au o poziție bună atunci când au o deschidere cât mai mare, spre exemplu când turnul este pe o coloană liberă. De asemenea, controlul asupra centrului tablei de șah este un alt aspect ce oferă avantaj unui dintre jucători. Pionii ce se află pe aceeași coloană dezavantajează jucătorul, deoarece se blochează unul pe celălalt. Pionii izolați nu se pot apăra între ei, aspect ce oferă adversarului oportunitatea de a atacă acești pioni.

Poate cel mai important criteriu ce trebuie evaluat este siguranță regelui pe tablă de șah, deoarece dacă regele se află într-o poziție vulnerabilă, există riscul să primească șah mat.

Domeniul funcției de evaluare este $[-\infty,\infty]$, astfel încât valorile pozitive reprezintă avantajul albului, iar valorile negative reprezintă avantajul negrului. Cu cât rezultatul evaluării are o valoare mai mare, cu atât este mai mare avantajul albului față de negru. Spre exemplu, dacă albul a dat șah mat jucătorului negru, poziția pe tablă de șah va fi evaluată la valoarea ∞, iar dacă pe o altă poziție pe tablă de șah albul are avantaj de 3 pioni față de negru, valoarea funcției de evaluare va fi mai mică.

Această funcție se mai numește și funcție de evaluare euristica, doarece nu este garantat că valoarea ei este cea optimala, ci oferă doar o aproximare a acestei valori.

3.1.2.2. Machine learning

Această abordare se potrivește în special pentru sfârșitul jocurilor de șah, deoarece pozițiile de final sunt reprezentate de cele mai multe ori de puține piese și prin urmare puține mutări posibile.

Cea mai de succes abordare bazată pe machine learning a fost „Reinforcement learning-diferență temporala", în care sistemul nu evaluează o anumită poziție de șah, ci are abilitatea de a prezice evaluarea în viitorul apropiat, asigurând astfel o consistență temporală.

Unul dintre cele mai populare sisteme ce folosesc această abordare este „Tezaurul tablelor" denumit și TD-Gammon. Sistemul a fost antrenat folosind diferență temporală jucând meciuri de unul singur. După 1,5 milioane de meciuri jucate pentru antrenament, sistemul TD-Gammon a ajuns la nivelul de master în table.

3.1.2.3. Deep learning

Această abordare are că principiu de bază modalitatea în care un jucător real gândește următoarea mutare și anume: el își imaginează un anumit număr de mutări în avans, dar nu evaluează toate pozițiile (forță brută) ci doar acelea care sunt cele mai semnificative, astfel timpul de căutare scade semnificativ. Sistemele bazate pe deep learning au în spate o rețea neuronală care are capabilitatea de a detectă anumite țipare în jocul de șah. De obicei, un astfel de sistem folosește că suport de antrenament jucărea meciuri de unul singur, precum și seturi de date extrase din alte jocuri de șah. Seturile de date se concentrează în zonă pozițiile de șah echilibrate, deoarece statistic vorbind, acestea apăr în cele mai multe situații în timpul unui meci. Un astfel de set de date conține perechi de tipul poziție de șah – evaluare.

3.2. Sisteme similare

În continuare vor fi prezentate sisteme similare cu ChessEngine. Unul dintre aceste sisteme este o aplicație web iar cealaltă aplicație este de șine stătătoare. Aceste sisteme au fost alese cu scopul de comparație, având mai multe caracteristici în comun cu sistemul ce este propus spre dezvoltare în lucrarea de față.

3.2.1. ChessMaster

ChessMaster este un joc de șah ce este distribuită că aplicație de șine stătătoare. În acest sens, jucătorul poate jucă împotrivă sistemului, având la dispoziție mai multe nivele de dificultate, poate vizualiză statistici, sau poate participă în turnee virtuale de șah. Conform Wikipedia, jucătorul virtual de șah ce stă la bază ChessMaster se numește „The King” și a fost implementat de olandezul Johan de Koning, fiind introdus încă din distribuția ChessMaster 4000. O funcționalitate interesantă a acestui sistem este aceea că le permite utilizatorilor să personalizeze stilul de joc al jucătorului virtual, prin configurarea unor setări precum siguranță regelui, mobilitatea pieselor, sau avantajul material [5].

3.2.2. Chess.com

Chess.com este o platformă destinată comunității jucătorilor de șah unde utilizatorii pot jucă meciuri între ei, pot jucă împotrivă sistemului, sau pot rezolvă probleme de șah. O funcționalitate interesantă a platformei este aceea că se pot jucă meciuri în echipă împotrivă sistemului, în sensul că sunt mai mulți jucători ce votează următoarea cea mai bună mutare. Mutarea care a primit cele mai multe voturi este aleasă că fiind mutarea câștigatoare. Funcționalități suplimentare ale platformei sunt forumul jucătorilor de șah, unde aceștia pot purta discuții și pot primi răspunsuri la întrebări din partea unor jucători experimentați, sau blogul chess.com, unde se postează articole despre șah.

3.2.3. Comparație

În continuare este prezentată o comparație între sistemul ChessEngine și celelalte două aplicații similare: Chess.com și ChessMaster. Criteriile alese pentru comparație au fost alese pentru a evidenția funcționalitățile de bază ale fiecărei platforme.

Tabel 3.1 Comparatie sisteme similare

Criteriu de evaluare	**ChessEngine**	**Chess.com**	**ChessMaster**
Aplicație WEB	✔	✔	✖
Permite interacțiune cu administratorul direct din aplicație	✔	✔	✖
Oferă tuoriale pentru jocul de șah	✔	✔	✔
Oferă un set e strategii de joc	✔	✔	✔
Interfață intuitivă	✔	✔	✔
Interfață simplificată, orientată către funcționalitatea esențială	✔	✖	✔
Oferă un top al jucătorilor	✔	✔	✖
Permite vizualizare de statistici personale	✔	✔	✔
Performanța sistemului nu este dependentă de stația jucătorului	✔	✔	✖
Permite modul multiplayer	✖	✔	✔
Permite diferite nivele de dificultate	✔	✔	✔

După cum se poate observă din tabel, diferență majoră între ChessEngine și celelalte aplicații este că ChessEngine nu permite modul multiplayer, cel puțin nu în versiunea curentă. Această îmbunătățire este specificată în capitolul „Dezvoltari ulterioare”.

Capitolul 4. Analiză şi Fundamentare Teoretică

În continuare vor fi prezentate aspecte legate de tehnologiile folosite și cazurile de utilizare ce se propun a fi implementate.

4.1. Tehnologii folosite

Dezvoltarea aplicației ChessEngine presupune utilizarea unui set de tehnologii ce oferă suport pentru partea de aplicație web, cum ar fi Spring, Hibernate sau MySQL. De asemenea, există un set de tehnologii folosite pe partea de interfață grafica, cu scopul de a ușura dezvoltarea aplicației, cum ar fi de exemplu librăria de JavaScript chessboard.js, ce oferă funcționalitatea necesară afișării și manipulării pieselor de șah pe tablă.

4.1.1. Tehnologii pe partea de server

4.1.1.1. Java

Java este un limbaj de programare obiect orientat, independent de platformă pe care rulează. Codul Java este compilat într-un limbaj intermediar, numit bytecode, ce este rulat mai departe de un Java Virtual Machine. Acest Java Virtual Machine convertește bytecode-ul în codul mașină. Java este unul dintre cele mai folosite limbaje pentru aplicațiile ce folosesc arhitectură client-server [17].

S-a optat pentru acest limbaj de programare, deoarece există multe tool-uri ce facilitează dezvoltarea codului, există o vastă documentație pe internet pentru Java și nu în ultimul rând, există multe framework-uri şi librării ce pot folosite în sprijinul dezvoltării unei aplicații de tip web. Exemple de astfel de framework-uri și librării sunt următoarele: Spring, Hibernate.

4.1.1.2. Tomcat

Este o implementare a specificației Java servlet. Tomcat este un server web dezvoltat de Apache Software Foundation care execută servlet-uri de java și randeaza pagini web care include JSP.

O alternativă la Tomcat este Jetty, un server web ce oferă suport pentru deploymentul aplicațiilor Java. Acest server este folosit în special în comunicarea între mașini.

S-a optat pentru Tomcat, deoarece oferă suport pentru deploy-ul aplicațiilor scrise în java și arhivate în structuri de tip war.

4.1.1.3. Spring

Este un framework de dezvoltare a aplicațiilor pentru enterprise java. Spring oferă suport pentru ciclul de viață al obiectelor, injecție de dependințe, funcționalități ce ajută programatorul să se focuseze mai mult pe business-ul aplicației [1], [2].

Cu ajutorul injecției de dependințe oferit de Spring, programatorul are posibilitatea să asambleze componentele între ele.

4.1.1.3.1. Facilități oferite de Spring

Injecția de dependințe

Orice aplicație bazată pe limbajul de programare java are obiecte care lucrează împreună pentru a oferi o funcționalitate. Pentru aplicațiile complexe este important că obiectele să fie cât mai independente cu putință pentru a creă posibilitatea refolosirii lor. Injecția de dependințe oferită de Spring oferă suport în acest sens. [13]

Există două tipuri de injecție de dependințe:

- Injecție de dependințe prin constructor (atunci când container-ul invocă un constructor de clasă cu un număr de argumente, fiecare reprezentând o dependință către o altă clasă)
- Injecție de dependințe prin settere (atunci când container-ul invocă un setter unui obiect având că argument o dependință către o altă clasă.

Inversion of Control (IoC)

Suportul oferit de Spring pentru inversion of control se aplică la management-ul obiectelor java și al ciclului de viață al acestora. Obiectele java ce sunt instantiate de containerul Ioc se numesc bean-uri. Container-ul de Ioc forțează modelul de injecție a dependințelor pentru obiectele java, rezultând un sistem decuplat.

Că și alternativă principală la Spring, din punct de vedere al injecției de dependințe, există Dagger. Dagger este o librărie ce poate fi folosită în aplicații mici, pennru a evită un număr mare de dependințe. Această librărie se folosește în special pentru aplicațiile mobile, datorită dimensiunilor mici ale acesteia.

O altă alternativă la Spring este Google Guice, un framework ce oferă suport pentru injecția de dependințe prin folosirrea anotatiilor în configuarea obiectelor Java. Google Guice oferă posibilitatea de a legă o clasă de interfață ei, iar când este nevoie de o altă implementare a interfeței, programatorul poate creă o nouă anotare cu ajutorul căreia să identifice nouă implementare.

S-a optat pentru Spring, deoarece aveam nevoie de suport pentru organizarea și legarea componentelor între ele. Fiind o arhitectură de tip layered, fiecare layer trebuie legat de layerul precedent. Spring a ajutat la simplificarea arhitecturii aplicației.

4.1.1.4. Spring MVC

Spring MVC este componența WEB a framework-lui. Spring MVC oferă o funcționalitate bogată pentru crearea aplicațiilor WEB robuste. Această componența este proiectată pentru a putea fi configurată cât mai ușor. Spring MVC este construit pe bază arhitecturii Model-View-Controller, ceea ce reprezintă o repartizare a claselor java în funcție de responsabilitatea acestora [18].

Spring MVC oferă suport pentru procesarea request-urilor HTTP ce vin de la client, prin manipularea formularelor transmise și afișarea informațiilor prezente pe partea de server într-un mod organizat.

O alternativă la Spring MVC este Grails, un framework ce oferă suport pentru dezvoltarea aplicațiilor web în Java. Principalela caracteristică a frameworkului Grails este că nu necesită configurare XML, în schimb Grails folosește un set de convenții, cum ar fi spre exemplu, fiecare clasă de tip Controller trebuie să aibă numele terminat cu “Controller”.

O altă alternativă este Vaadin. Acest framework oferă suport pentru dezvoltarea aplicațiilor web în Java și se bazează pe o arhitectură orientată pe evenimente. Vaadin folosește Google Web Toolkit pentru randarea pagînilor web. Vaadin vine cu un set de

componente web deja făcute, fiecare având și partea de client și cea de server implementată.

S-a optat pentru Spring MVC deoarece oferă o separare clară a responsabilităților între controllere, modele și view-uri, controllerele pot fi configurate folosing inversion of control, la fel că și celelalte componente folosite în aplicație. Astfel, în aplicația de față controllerele procesează request-urile HTTP, view-urile randeaza interfață grafica a aplicației, iar modelul este responsabil cu transmiterea informației.

4.1.1.5. Spring security

pring security este un framework ce oferă suport pentru management-ul utilizatorilor: autentificarea și autorizarea acestora pentru a vizualiză anumite pagini ale aplicației [20].

Autentificarea unui utilizator este mecanismul prin care sistemul identifică utilizatorii.

Autorizarea este un mecanism prin care sistemul determină rolul utilizatorului și nivelul de acces la anumite pagini ale aplicației.

O alernativa la Spring security este Apache Shiro, un framework ce oferă suport pentru implementarea proceselor de autentificare și autorizare în aplicațiile Java bazate pe Spring. Apache Shiro este un framework simplificat, având multe posibilități de configurare.

S-a optat pentru Spring Security, deoarece oferă suport pentru management-ul utilizatorilor și se integrează cu framework-ul principal folosit în aplicație, și anume Spring.

4.1.1.6. MySQL

"MySQL este un sistem de gestiune a bazelor de date relaționale, produs de compania suedeză MySQL AB și distribuit sub Licență Publică Generală GNU. Este cel mai popular SGBD open-source la oră actuala".

MySQL este un RDBMS (relațional database management system) folosit pentru dezolvtare aplicațiilor web.

Bazele de date relaționale sunt folosite pentru a stoca și organiza volume meri de date. Termenul de bază de date relațională vine de la faptul că datele sunt stocate în tabele diferite între case sunt stabilite relații utilizând chei primare și chei straine". [12]

O alternativă la MySQL este PostGreSQL, un sistem de baze de date relaționale. Dintre funcționalitățile sale, iese în evidența executorul eficient atât pentru interogările statice, cât și pentur cele parametrizate. TOAST compression, sau impoved cache management. De asemenea, oferă un grad ridicat de scalabilitate în situațiile cu un număr ridicat de scrieri în bază de date.

O altă alternativă la MySQL sunt sistemele de gestiune a bazelor de date NoSQL, acestea făcând un compromis asupra consistenței datelor în favoarea disponibilității. În comparație cu bazele de date relaționale, NoSQL sunt mai scalabile și oferă o performanță superioară, prin posibilitatea folosirii unei arhitecturi distribuite, în locul uneia monolitice. Exemple de baze de date NoSQL sunt bazele de date bazate pe documente (MongoDB), pe grafice (Neo4j), pe perechi de tipul cheie-valoare (Riak).

S-a optat pentru MySQL, deoarece:

- Este ușor de utilizat, fiind necesar un număr minim de pași pentru instalarea și utilizarea acestui sistem de gestiune
- Este securizat: există mai multe tipuri de utilizatori cu diferite roluri, având atribuite diferite drepturi.
- Este rapid
- Este scalabil

4.1.1.7. Hibernate

Hibernate este un serviciu de persistență obiect-relațional, un ORM (object relațional mapping) cu responsabilitatea principală de a mapa clasele java la tabelele din bază de date [11].

Object Relațional Mapping este o tehnică de programare pentru convertirea datelor între bazele de date relaționale și limbajele de programare obiect orintate.

Principalul avantaj al ORM-urilor este acela că oferă o interfață de nivel mai înalt bazei de date decât JDBC, ascunzând detaliile interogărilor SQL.

Un alt avantaj important este că ORM-urile separă aplicația de bază de date, ceea ce oferă posibilitatea schimbării sistemului de gestiune a bazei de date cu ușurință.

O alternativă la Hibernate este OJB (ObjectRelationalBridge), un framework ce oferă suport pentru persistarea obiectelor java într-un mod transparent.

S-a optat pentru Hibernate în primul rând deoarece este independent de bază de date folosită și oferă o interfață de nivel înalt asupra bazei de date.

4.1.1.8. Junit

Junit este un framework utilizat pentru testarea aplicațiilor scrise în limbajul de programare java.

Principalul avantaj al Junit este că testele pot fi rulate automat. Un alt avantaj este că testele pot fi organizate atât sub formă de suite de test,

S-a optat pentru Junit pentru testarea corectitudinii algoritmilor utilizați în aplicație.

4.1.1.9. Jackson

Jackson este o bibliotecă ce permite serializarea și deserializarea obiectelor java în JSON. Principalul avantaj acestei librării este că nu e nevoie de o mapare manuală între obiectul java și json. Un alt avantaj constă în faptul că această librărie nu necesită dependințe suplimentare.

S-a optat pentru Jackson deoarece permite serializarea și deserializarea obiectelor Java într-un mod transparent și ușor de folosit.

4.1.1.10. Maven

Maven este un sistem de build și management al proiectelor, scris în Java. Funcționalitățile sale principale sunt descrierea procesului de build al softwareului și descrierea dependențelor acestuia. Maven descărcă automa bibliotecila neceare dintr-un repository online.

O alternativă la Maven este Gradle, un framework pentru buildul aplicației, ce folosește un limbaj bazat pe Groovy. Gradle este proiectat pentru build-uirea mai multor

proiecte care pot să crească exponențial și suportă build-uri incrementale, identificând în mod inteligent care părți ale build-ului precedent sunt actuale și ce părți s-au schimbat.

O altă alternativă la Maven este Apache Ant, un tool pentru build-ul aplicației, ce folosește XML că și limbaj de descriere.

S-a optat pentru Maven deoarece oferă suport pentru managementul dependințelor, descărcând automat bibliotecile necesare. Astfel am obținut un mai bun control asupra versiunilor bibliotecilor și dependințelor dintre modulele aplicatiei.

4.1.1.11. JSP

JavaServer Pages este o tehnologie de programare pe partea de server ce permite crearea de pagini WEB dinamice și independente de platformă [16].

S-a optat pentru JSP deoarece este tehnologia recomandată de randare a informațiilor ce vin din controllerele create cu sprijinul framework-ului Spring MVC.

4.1.2. Tehnologii pe partea de client

4.1.2.1. HTML

HTML este un limbaj de marcare utilizat pentru crearea pagînilor web ce pot fi afișate într-un browser. Scopul HTML este acela de a prezența informațiile prin organizarea lor în paragrafe, tabele, liste sau link-uri.

S-a folosit HTML pentru organizarea informațiilor în paginile interfeței grafice.

4.1.2.2. CSS

Este un standard pentru formatarea elementelor unui document HTML. Se pot atașa diferite stiluri tag-urilor de HTML, astfel oferind o experiență mai plăcută utilizatorului.

S-a folosit CSS pentru a formată informațiile din paginile interfeței grafice într-un mod intuitiv, cu focalizare asupra informațiilor importante.

4.1.2.3. JavaScript

Este un limbaj de programare obiect orientat bazat pe conceptul prototipurilor. Este folosit mai ales pentru introducerea unor funcționalități în paginile web, codul javascript din aceste pagini fiind rulat de către browser.

Utilizarea Javascriptului este după cum spun cei de la Wikipedia [14]:

“Cea mai des intalnită utilizare a JavaScript este în scriptarea pagînilor web. Programatorii web pot îngloba în paginile HTML script-uri pentru diverse activitări cum ar fi verificarea datelor introduse de utilizatori sau crearea de meniuri și alte efecte animate.

Browserele rețin în memorie o reprezentare a unei pagini web sub formă unui arbore de obiecte și pun la dispoziție aceste obiecte script-urilor JavaScript, care le pot cîți și manipulă. Arborele de obiecte poartă numele de Document Object Model sau DOM. Există un standard W3C pentru DOM-ul pe care trebuie să îl pună la dispoziție un browser, ceea ce oferă premiza scrierii de script-uri portabile, care să funcționeze pe toate browserele. În practică, insă, standardul W3C pentru DOM este incomplet implementat.

Deși tendința browserelor este de a se alinia standardului W3C, unele din acestea incă prezintă incompatibilități majore, cum este cazul Internet Explorer.”

Principalul motiv pentru care s-a optat pentru Javascript este pentru a putea manipulă tablă de șah.

4.1.2.4. Jquery

Jquery este o platformă de dezvoltare bazată pe Javascript, concepută pentru a ușura și îmbunătăți procese precum modificare conținutului paginii în mod dinamic, selectarea informației din pagină sau animații grafice.

S-a folosit Jquery pentru navigarea între mutările unei liste de strategii de șah.

4.1.2.5. Ajax

Este prescurtarea de la Asynchronous Javascript and XML, este o tehnică de programare pentru crearea de aplicații web interactive ce permite transmiterea de request-uri HTTP asyncrone.

S-a folosit Ajax pentru a genera următoarea mutare în cadrul unul meci de șah.

4.1.2.6. ChessBoard.js

Este un API ce randeaza tabla de șah, fiind scris in JavaScript [15].

S-a folosit acest API pentru randarea tablei de șah.

4.1.3. Tehnologii pentru comunicarea între client si server

4.1.3.1. HTTP

Este un protocol de comunicare bazat pe TCP/IP care este folosit pentru a transmite date in format text, HTML, Json. Dintre avantajele protocolului de comunicare HTTP se remarcă urmatoarele:

- HTTP nu necesită o conexiune: clientul HTTP initiaăa un request HTTP iar după ce request-ul este trimis, clientul se deconectează de la server si așteaptă raspunsul HTTP. Server-ul proceseaza request-ul ți reinițiază conextiunea cu clientul, trimițând răspunsul HTTP inapoi.
- Media-independent: orice tip de date poate fi transferat utilizand HTTP, atât timp cât și clientul și serverul sunt capabili să utilizeze acel tip de date.
- HTTP nu are stare

S-a folosit HTTP deoarece acesta este protocolul implicit de comunicare în aplicațiile WEB bazate pe Spring MVC.

4.1.3.2. JSON

Este un format de prezentare și interschimb de date între aplicații. Principalul avantaj al JSON-ului este acela că nu depinde de limbajul în care e scrisă aplicația ce trimite sau ceea ce primește informația.

S-a folosit JSON deoarece necesită mai puține metadate decât XML.

4.2. Cerințele sistemului

În continuare vor fi descrise cerințele sistemului. Cerințele funcționale descriu comportamentul anumitor componente, iar cerințele non-funcționale se referă la anumiți parametrii de funcționare ai sistemului în general, spre exemplu: utilizabilitate, performanță, sau securitate.

4.2.1. Cerințe funcționale

Cerințele funcționale definesc comportamentul sistemului în anumite situații, precum și componentele pe care utilizatorii le vor folosi în interacțiunea cu sistemul. În continuare este prezentat un tabel cu cerințele funcționale ale sistemului:

Tabel 4.1 Cerințe funcționale

Identificator	Descrierea cerinței funcționale	Utilizator care beneficiază de funcționalitate
CF 1.0	Meci de șah împotriva jucătorului virtual	Jucător autentificat și jucător neautentificat
CF 1.1	Managementul meciurilor de șah (salvare meci de șah, revenire la meci de șah)	Jucător autentificat
CF 2.0	Probleme de șah	Jucător autentificat și jucător neautentificat
CF 2.1	Problema de șah a zilei	Jucător autentificat și jucător neautentificat
CF 3.0	Strategii de șah	Jucător autentificat și jucător neautentificat
CF 4.0	Ultimele noutăți	Jucător autentificat și jucător neautentificat
CF 5.0	Citatul zilei	Jucător autentificat și jucător neautentificat
CF 6.0	Trimitere mesaj către administrator	Jucător autentificat și jucător neautentificat
CF 7.0	Vizualizare statistici	Administratorul
CF 8.0	Managementul utilizatorilor	Administratorul
CF 8.1	Managementul mesajelor	Administratorul
CF 8.2	Managementul strategiilor de șah	Administratorul
CF 8.3	Managementul problemelor de șah	Administratorul
CF 8.4	Managementul citatelor	Administratorul

4.2.2. Cerințe nonfuncționale

Cerințele nonfunctionale descriu atribute ale sistemului precum utilizabilitatea acestuia, performanță sau suportabilitatea. Aceste atribute pot fi văzute totodată că și constrângeri sau restricții asupra proiectării sistemului (e.g. din punct de vedere al performanței, sistemul trebuie să aibă un timp de răspuns mai mic de o secundă).

4.2.2.1. Utilizabilitatea

Utilizabilitatea unui sistem este dată de ușurință cu care acesta poate fi utilizat și învățat. Deoarece această aplicație va avea și utilizatori neexperimentați, așa numiții jucători ocazionali de șah, ușurință în utilizare joacă un rol important în aplicație. Interfață grafica trebuie să fie intuitivă, iar utilizatorii să o poată folosi fără să fie nevoiți să apeleze la manualul de utilizare. Funcționalitățile principale trebuie să fie accesibile din orice punct al aplicației prin realizarea unui număr minim de pași. Utilizatorii trebuie să știe în orice moment în ce stare se află sistemul: jucătorul trebuie să facă o mutare în cadrul unui meci de șah, sistemul trebuie să facă o mutare, meciul este oprit, utilizatorul este autentificat, sau nu este autentificat.

Un alt aspect important legat de utilizabilitatea sistemului este simplitatea interfeței grafice. Aceasta trebuie să fie focusată către informația utilă, fiind evitate formele grafice ce ar atrage atenția utilizatorului.

4.2.2.2. Performanța

Performanța se referă la timpul de răspuns necesar sistemului pentru a procesa o cerere din partea utilizatorului. Cum aplicația ChessEngine are două categorii de cereri: cereri ce implică activitatea jucătorului virtual de șah și cereri ce nu implică activitatea jucătorului virtual de șah, se vor defini următorii timpi de răspuns:

- Pentru cererile ce implică activitatea jucătorului de șah, timpul de răspuns trebuie să fie sub 5 secunde. Un exemplu de cerere ce implică activitatea jucătorului virtual de șah este generarea mutării următoare în cadrul unui meci de șah.
- Pentru cererile ce nu implică activitatea jucătorului de șah, timpul de răspuns trebuie să fie sub o secundă. Un exemplu de cerere ce nu implică activitatea jucătorului virtual de șah este încărcarea unei pagini din interfața grafică.

4.2.2.3. Suportabilitatea

Suportabilitatea reprezintă abilitatea sistemului de a fi modificat cu ușurință și totodată ușor mentenabil. Fiind conceput pe o arhitectură de client-server, interfață grafica este decuplată față de logică de business. De asemenea, în cadrul serverului, jucătorul virtual de șah este decuplat de restul serviciilor printr-o interfață. La fel și bază de date, este decuplată față de sistem, lucru ce permite schimbarea bazei de date folosite cu ușurință.

4.2.2.4. Securitatea

Securitatea este abilitatea sistemului de a restricționa accesul la anumite pagini din interfață grafica. În cazul sistemului chessEngine, zonă de administrare trebuie securizată prin restricționarea accesului tuturor utilizatorilor obișnuiți și autorizarea unuia singur: administratorul. Paginile ce au scopul de administrare vor putea fi accesate doar de către administrator, prin autentificare.

4.3. Cazuri de utilizare

4.3.1. Actorii sistemului

Pentru interacțiunea cu sistemul,există 3 tipuri de utilizatori: jucătorul neautentificat, jucătorul autentificat și administratorul. În continuare sunt descrise drepturile și funcționalitățile puse la dispoziție pentru fiecare tip de utilizator.

4.3.1.1.Jucător neautentificat

Jucătorul neautentificat este acel tip de utilizator ce accesează pentru prima data aplicația sau dorește să folosească doar funcționalitățile de bază puse la dispoziție: jucărea unui meci de șah, vizualizare de strategii sau rezolvarea problemelor de șah. Acest tip de utilizator nu are un cont asignat.

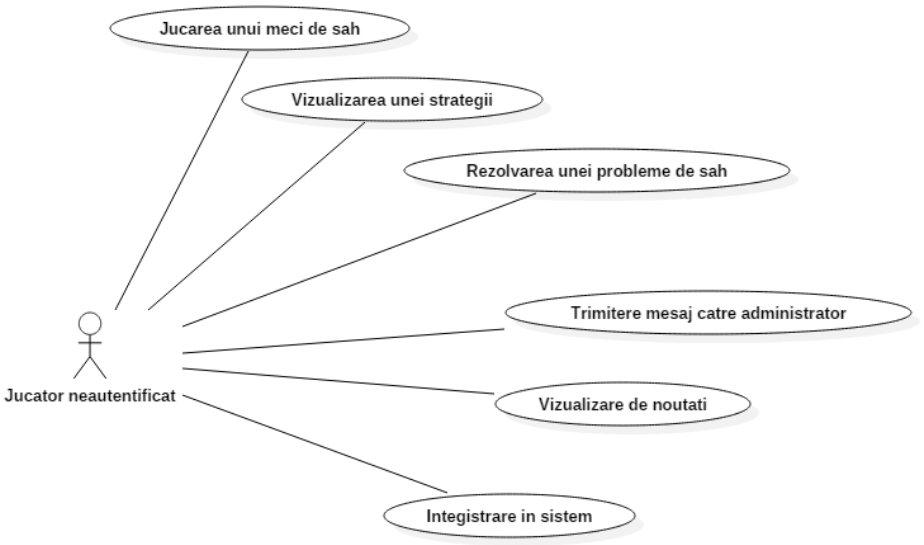

Figura 4.1 Cazuri de utilizare pentru jucătorul neautentificat

4.3.1.2.Jucător autentificat

Jucătorul autentificat este acel tip de utilizator ce accesează frecvent aplicația și dorește să beneficieze de funcționalități adiționale față de cele ale jucătorului neautentificat, precum: salvarea unui meci de șah, reluarea unui meci salvat sau vizualizarea de statistici. Pentru a deveni jucător autentificat, utilizatorul trebuie să se înregistreze în sistem prin completarea unui formular de înregistrare. Pentru a accesă funcționalitățile suplimentare, utilizatorul trebuie să se autentifice în sistem.

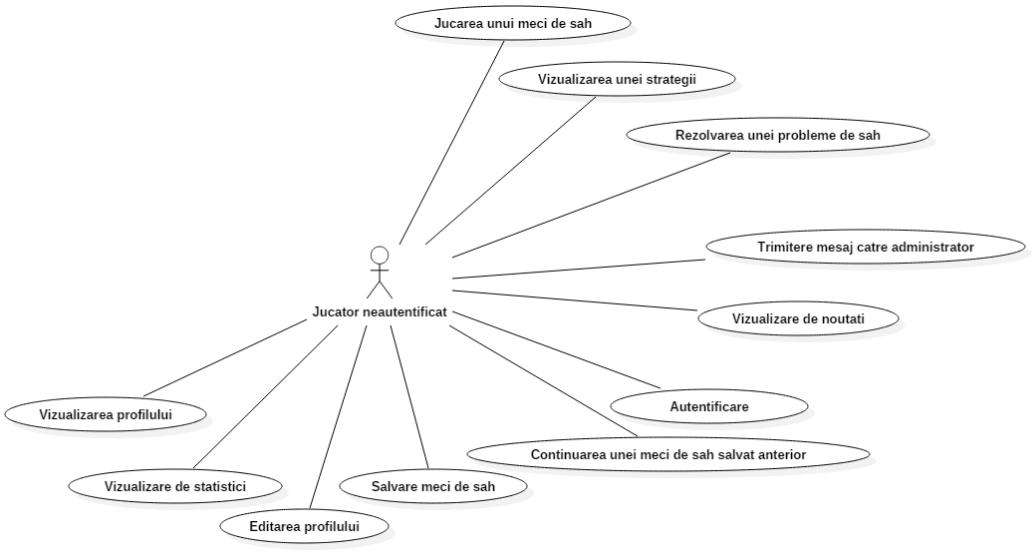

Figura 4.2 Cazuri de utilizare pentru jucătorul autentificat

4.3.1.3. Administrator

Administratorul este acel tip de utilizator responsabil cu managementul aplicației: el are la dispoziție o interfață grafica separată unde poate vedea lista de utilizatori existenți și are dreptul să blocheze anumiți utilizatori. De asemenea, tot prin intermediul acestei interfețe administratorul poate vedea mesajele primite, sau poate adaugă strategii și probleme noi de șah.

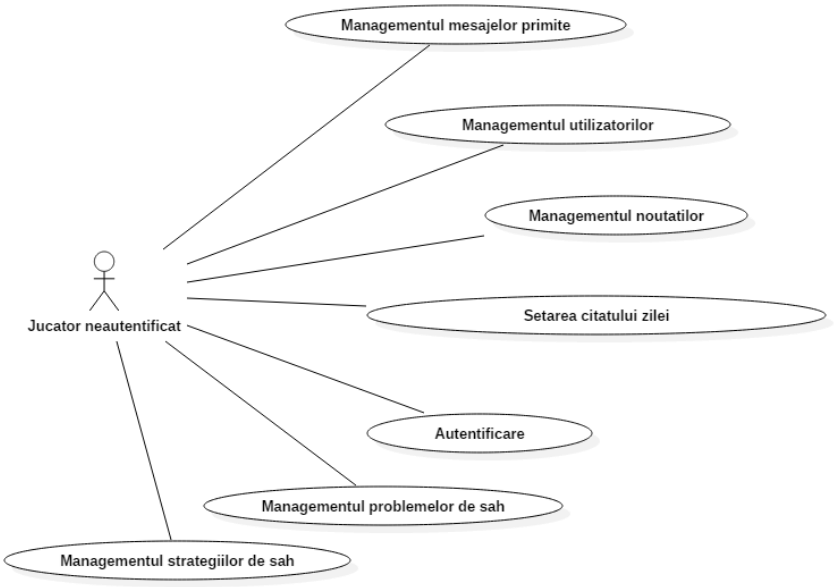

Figura 4.3 Cazuri de utilizare pentru administrator

4.3.2. Cazuri de utilizare pentru jucătorii autentificați si cei neautentificați

4.3.2.1. Cazul de utilizare 1

Numele cazului de utilizare: Jucarea unui meci de șah
Actor principal: Jucătorul autentificat sau neautentificat
Părți interesate (Stakeholders) :

- Administratorul: dorește sa aibă o statistică centralizată cu meciurile jucate

Precondiții: Jucătorul se află pe o pagină a aplicației ce afisează meniul cu cercuri.

Postcondiții: Meciul a fost terminat, salvat sau a fost revocat.
Scenariul de succes:

1. Jucătorul accesează opțiunea pentru jucarea unui meci de șah
2. Sistemul afisează o pagină intermediară cu setările meciului
3. Jucătorul selectează culoarea pieselor cu care dorește să joace și nivelul jucătorului virtual de șah
4. Sistemul redirectionează jucătorul de șah la tabla de șah.
5. Jucătorul face o mutare
6. Sistemul răspunde cu o altă mutare.
7. Acest ciclu se repetă până când unul dintre jucatori învinge
8. Sistemul actualizează profilul jucatorului cu rezultatul noului meci.

Scenarii alternative:

5a Utilizatorul încearcă să închidă accidental tab-ul din browser în care se desfășoară meciul sau navighează către altă pagină:

1. Sistemul afișează un mesaj de confirmare a recovării meciului.
2. Utilizatorul confirmă revocarea meciului sau renunță la revocarea meciului.

4a Utilizatorul încearcă să facă o mutare invalidă cu propriile piese:

1. Sistemul reașează piesa mutată în poziția în care aceasta se afla înaintea mutării.

4b Utilizatorul încearcă să facă o mutare cu piesele adversarului:

1. Sistemul nu permite mutarea unei piese a adversarului.

4.3.2.2. Cazul de utilizare 2

Numele cazului de utilizare: Salvarea unui meci de șah
Actor principal: Jucătorul autentificat
Părți interesate (Stakeholders) :

- Administratorul: dorește să aibă o statistică centralizată cu meciurile jucate

Precondiții: Jucătorul autentificat se află în timpul unui meci de șah.
Postcondiții: Meciul a fost salvat și se găseste în lista de meciuri salvate.
Scenariul de success:

1. Jucătorul selectează opțiunea de salvare a meciului
2. Sistemul afisează un popup pentru introducerea numelui meciului

3. Jucătorul introduce numele meciului și confirmă acțiunea de salvare a acestuia.

Scenarii alternative:
2a Jucătorul dorește să continue meciul:
1. Jucătorul selectează opțiunea de revocare a salvării meciului și continuă să joace.

4.3.2.3. Cazul de utilizare 3

Numele cazului de utilizare: Reluarea unui meci salvat de șah
Actor principal: Jucătorul autentificat
Precondiții: Jucătorul autentificat se află pe orice pagină a interfeței grafice.
Postcondiții: Meciul de șah este reluat de la poziția la care a fost salvat.
Scenariul de success:
1. Jucătorul selectează opțiunea de reluare a unui meci salvat de șah din meniu.
2. Sistemul redirecționează utilizatorul către o pagină în care sunt afișate meciurile de șah salvate.
3. Utilizatorul selectează un meci.
4. Sistemul redirecționează utilizatorul către meciul de șah dorit si setează poziția pieselor în mod respunzător pe tabla de șah.

Scenarii alternative:
3a Meciul salvat nu poate fi încarcat:
1. Sistemul afisează un mesaj de eroare pe ecran.

4.3.2.4. Cazul de utilizare 4

Numele cazului de utilizare: Vizualizarea unei strategii de șah
Actor principal: Jucătorul autentificat sau neautentificat
Precondiții: Jucătorul se află pe o pagină a aplicației ce afisează meniul cu cercuri.
Postcondiții: Utilizatorul vizualizează o strategie de șah.
Scenariul de success:
1. Jucătorul selectează opțiunea de vizualizare a unei strategii de șah meniu.
2. Sistemul redirecționează utilizatorul către o pagină în care este afișată lista de strategii.
3. Utilizatorul selectează o strategie pe care dorește să o vizualizeze.
4. Sistemul redirecționează utilizatorul către o pagină în care este afișată tabla de șah și lista de mutări din strategie.
5. Utilizatorul selectează o mutare din listă.
6. Sistemul setează pe tabla de șah poziția corespunzătoare mutării selectate.

Scenarii alternative:
3a Strategia selectată nu poate fi incarcată:
1. Sistemul afisează un mesaj de eroare pe ecran.

4.3.2.5. Cazul de utilizare 5

Numele cazului de utilizare: Rezolvarea unei probleme de șah
Actor principal: Jucătorul autentificat sau neautentificat
Precondiții: Jucătorul se află pe o pagina a aplicației ce afisează meniul cu cercuri.
Postcondiții: Utilizatorul rezolvă o problema de șah.
Scenariul de success:

1. Jucătorul selectează opțiunea de vizualizare a problemelor de șah meniu.
2. Sistemul redirecționează utilizatorul către o pagină în care este afișata lista de probleme disponibile.
3. Utilizatorul selectează o problemă pe care dorește să o rezolve.
4. Sistemul redirecționează utilizatorul către o pagina in care este afisata table de șah si cerinta problemei.
5. Utilizatorul face o mutare.
6. Sistemul răspunde cu o mutare.
7. Pasii 5 si 6 se repeta pana cand se ajunge la rezolvarea problemei de șah.
8. Sistemul afiseaza un mesaj corespunzator rezolvarii cu succes a problemei de șah.

Scenarii alternative:
3a Problema de șah nu poate fi încarcată:

1. Sistemul afiseaza un mesaj de eroare pe ecran.

5a Utilizatorul face o mutare gresita:

1. Sistemul afiseaza un mesaj prin care instiinteaza utilizatorul ca a facut o mutare gresita si muta piesa in pozitia in care se afla inainte de mutare.

4.3.2.6. Cazul de utilizare 6

Numele cazului de utilizare: Autentificarea jucatorului în sistem
Actor principal: Jucătorul neautentificat
Precondiții: Jucătorul neautentificat se află pe o pagina a aplicației.
Postcondiții: Jucătorul este autentificat în sistem.
Scenariul de success:

1. Jucătorul neautentificat selectează opțiunea de autentificare in sistem.
2. Sistemul redirecționează utilizatorul pe o pagina pe care se afla formularul de autentificare.
3. Utilizatorul introduce numele și parola și pornește autentificarea în sistem.
4. Sistemul verifică dacă datele introduse sunt valide si redirecționează utilizatorul la pagina pe care era anterior, afișând în partea de sus a paginii numele utilizatorului autentificat.

Scenarii alternative:
3a Utilizatorul introduce date invalide:

1. Sistemul redirecționează utilizatorul la formularul de autentificare și afisează un mesaj de eroare pe ecran.

4.3.2.7. Cazul de utilizare 7

Numele cazului de utilizare: Înregistrarea unui utilizator nou
Actor principal: Jucătorul neautentificat
Precondiții: Jucătorul neautentificat se află pe pagina cu formularul de autentificare în sistem
Postcondiții: Este creat un cont nou pentru utilizator
Scenariul de success:

1. Jucătorul selectează opțiunea de înregistrare.
2. Sistemul redirecționează jucătorul la o pagină cu un formular de înregistrare.
3. Jucătorul completează formularul de înregistrare și trimite formularul.

Scenarii alternative:
3a Utilizatorul introduce un nume de utilizator ce există deja:

1. Sistemul redirecționează utilizatorul la formularul de înregistrare și afisează un mesaj de eroare pe ecran.

3b Utilizatorul nu reintroduce corect parola aleasă:

1. Sistemul redirecționează utilizatorul la formularul de înregistrare și afisează un mesaj de eroare pe ecran.

4.3.2.8. Cazul de utilizare 8

Numele cazului de utilizare: Tentativa de a accesa o pagina cu acces restricționat la o anumită categorie de utilizatori
Actor principal: Jucătorul neautentificat sau autentificat
Precondiții: Jucătorul se află pe o pagina a interfeței grafice.
Postcondiții: Pagina cu acces restrictionat este protejată față de jucator.
Scenariul de success:

4. Jucătorul accesează o pagină cu acces restricționat.
5. Sistemul redirecționează utilizatorul la pagina de autentificare.

4.3.2.9. Cazul de utilizare 9

Numele cazului de utilizare: Vizualizarea profilului
Actor principal: Jucătorul autentificat
Precondiții: Jucatorul autentificat se află pe orice pagină a aplicației.
Postcondiții: Jucătorul vede informațiile profilului său.
Scenariul de success:

6. Jucătorul selectează opțiunea de vizualizare a profilului.
7. Sistemul redirecționează utilizatorul la pagina cu informațiile profilului.

4.3.2.10. Cazul de utilizare 10

Numele cazului de utilizare: Editarea profilului
Actor principal: Jucătorul autentificat
Precondiții: Jucătorul autentificat se afla pe orice pagină a aplicației.
Postcondiții: Jucătorul actualizează informațiile profilului său.
Scenariul de success:

1. Jucătorul selectează opțiunea de vizualizare a profilului.
2. Sistemul redirecționează utilizatorul la pagina cu informațiile profilului.
3. Utilizatorul selectează opțiunea de editare a profilului.
4. Sistemul afisează un formular de editare a profilului
5. Utilizatorul actualizează informațiile profilului și confirmă salvarea lor.
6. Sistemul actualizează informațiile și redirecționează utilizatorul la pagina de profil, unde acesta poate vizualiza informațiile actualizate.

Scenarii alternative:
6a Utilizatorul a introdus date invalide:

- Sistemul redirecționează utilizatorul la formularul de actualizare al profilului și evidențiază câmpurile invalide.

4.3.2.11. Cazul de utilizare 11

Numele cazului de utilizare: Vizualizarea de statistici
Actor principal: Jucătorul autentificat
Precondiții: Jucătorul autentificat se află pe orice pagină a aplicației.
Postcondiții: Jucătorul vede statisticile profilului său.
Scenariul de success:

1. Jucătorul selectează opțiunea de vizualizare de statistici din meniu.
2. Sistemul redirecționează utilizatorul la pagina cu statisticile, unde se află informații precum numărul de meciuri câștigate, pierdute sau terminate cu remiză.

4.3.2.12. Cazul de utilizare 12

Numele cazului de utilizare: Trimiterea unui mesaj
Actor principal: Jucătorul autentificat sau neautentificat
Precondiții: Jucătorul se afla pe orice pagina a aplicatiei.
Postcondiții: Jucătorul trimite un mesaj administratorului.
Scenariul de success:

1. Jucătorul selectează opțiunea de trimitere a unui mesaj din meniu.
2. Sistemul redirecționează utilizatorul la pagina de trimitere a unui mesaj, unde este afișat un formular ce are cîmpuri pentru datele de contact ale jucătorului și mesajul ce urmează a fi introdus.
3. Utilizatorul completează formularul prin introducerea datelor de contact și mesajul propriu-zis și confirmă trimiterea mesajului.

4. Sistemul receptionează mesajul și redirecționează jucatorul pe o pagină unde se află un mesaj de confirmare a primirii mesajului.

Scenarii alternative:
2a Jucătorul este autentificat în sistem:

Sistemul completează automat câmpul de introducere a datelor de contact pentru utilizator cu informațiile acestuia.

4a Utilizatorul a introdus date invalide:

Sistemul redirecționează utilizatorul înapoi la formular și evidențiază câmpurile ce conțin informații invalide.

4.3.3. Cazuri de utilizare pentru administrator

4.3.3.1. Cazul de utilizare 11

Numele cazului de utilizare: Accesarea panoului de administrare
Actor principal: Administratorul neautentificat
Precondiții: Administratorul neautentificat se află pe orice pagină a aplicației.
Postcondiții: Administratorul se află pe pagina principală de administrare.
Scenariul de success:

1. Administratorul neautentificat accesează link-ul pentru panoul de administrare.
2. Sistemul redirecționează administratorul către pagina de autentificare
3. Administratorul introducere numele și parola contului său și confirmă autentificarea
4. Sistemul validează autentificarea și redirecționează utilizatorul către pagina principală a panoului de administrare.

Scenarii alternative:
3a Administratorul a introdus datele de autentificare greșite:
- Sistemul redirecționează administratorul la formularul de autentificare și afișează un mesaj corespunzător situației.

4.3.3.2. Cazul de utilizare 12

Numele cazului de utilizare: Managementul noutăților
Actor principal: Administratorul
Precondiții: Administratorul se află pe orice pagină a panoului de administrare.
Postcondiții: Administratorul adaugă sau șterge o noutate.
Scenariul de success:

1. Administratorul accesează pagina de management a noutăților din meniu.
2. Sistemul redirecționează administratorul către pagina de management a noutăților.
3. Administratorul alege opțiunea de adaugare a unei noutăți.

4. Sistemul redirecționează utilizatorul către pagina de adaugare a unei noutăți.
5. Administratorul adauga o noutate.
6. Sistemul salveaza noutatea.

Scenarii alternative:
3a Administratorul alege opțiunea de ștergere a unei noutăți :

- Sistemul șterge noutatea și actualizează lista de noutăți.

4.3.3.3. Cazul de utilizare 13

Numele cazului de utilizare: Managementul citatelor
Actor principal: Administratorul
Precondiții: Administratorul se află pe orice pagină a panoului de administrare.
Postcondiții: Administratorul adaugă sau șterge un cititat sau selectează un citat ca fiind citatul zilei.
Scenariul de success:

1. Administratorul accesează pagina de management a citatelor din meniu.
2. Sistemul redirecționează administratorul către pagina de management a citatelor.
3. Administratorul alege opțiunea de adăugare a unui citat.
4. Sistemul redirecționează administratorul către pagina de adăugare a unui citat.
5. Administratorul adaugă un citat.
6. Sistemul salvează citatul.

Scenarii alternative:
3a Administratorul alege opțiunea de ștergere a unui citat:
- Sistemul șterge citatul și actualizează lista de citate.

3b Administratorul alege opțiunea de selectare a unui citat ca fiind citatul zilei
- Sistemul selectează citatul ca fiind citatul zilei.

4.3.3.4. Cazul de utilizare 14

Numele cazului de utilizare: Managementul problemelor de șah
Actor principal: Administratorul
Precondiții: Administratorul se află pe orice pagină a panoului de administrare.
Postcondiții: Administratorul adaugă sau șterge o problemă de șah sau selectează o problemă ca fiind problema zilei.
Scenariul de success:

1. Administratorul accesează pagina de management a problemelor de șah din meniu.
2. Sistemul redirecționează administratorul către pagina de management a problemelor.
3. Administratorul alege opțiunea de adăugare a unei probleme.

4. Sistemul redirecționează administratorul către pagina de adaugare a unei probleme.
5. Administratorul adaugă o cerință, o poziție inițială și o listă de mutări ce reprezintă soluția problemei.
6. Sistemul salvează problema.

Scenarii alternative:
3a Administratorul alege opțiunea de ștergere a unei probleme:
- Sistemul șterge problema și actualizează lista de probleme de șah.

3b Administratorul alege opțiunea de selectare a unei probleme ca fiind problema zilei
- Sistemul selectează problema ca fiind problema zilei.

4.3.3.5. Cazul de utilizare 15

Numele cazului de utilizare: Managementul utilizatorilor
Actor principal: Administratorul
Precondiții: Administratorul se află pe orice pagină a panoului de administrare.
Postcondiții: Administratorul vizualizează, blochează sau șterge un utilizator
Scenariul de success:
1. Administratorul accesează pagina de management utilizatorilor din meniu.
2. Sistemul redirecționează administratorul către pagina de management a utilizatorilor.
3. Administratorul alege opțiunea de vizualizare a unui utilizator
4. Sistemul redirecționează administratorul către profilul utilizatorului

Scenarii alternative:
3a Administratorul alege opțiunea de ștergere a unui utilizator:
- Sistemul șterge utilizatorul și actualizează lista de utilizatori.

3b Administratorul alege opțiunea de blocare a unui utilizator:
- Sistemul blochează utilizatorul.

4.3.3.6. Cazul de utilizare 16

Numele cazului de utilizare: Managementul mesajelor primite
Actor principal: Administratorul
Precondiții: Administratorul se află pe orice pagină a panoului de administrare.
Postcondiții: Administratorul vizualizează, sau șterge un mesaj.
Scenariul de success:
1. Administratorul accesează pagina de management mesajelor din meniu.
2. Sistemul redirecționează administratorul către pagina de management a mesajelor unde acestea sunt afișate sub formă de listă.

Scenarii alternative:

2a Administratorul alege opțiunea de ștergere a unui mesaj:
- Sistemul șterge mesajul si actualizeaza lista de mesaje.

Capitolul 5. Proiectare de Detaliu și Implementare

În acest capitol se va descrie proiectarea sistemului atât pentru partea de server, cât și pentru partea de client și anume interfață grafica. De asemenea, se va prezenta modul de implementare al jucătorului virtual de șah, structurile de date și algoritmul folosit. Pentru a ușura înțelegerea arhitecturii, se vor folosi diagrame UML.

5.1. Arhitectura sistemului

Arhitectură generală pentru care s-a optat este de aplicație web. Pe partea de backend a fost implementată logică aplicației, inclusiv jucătorul virtual de șah. Pentru această componența s-a folosit arhitectură mulți-tiered. Pe partea de front-end s-a plasat interfață grafica. Comunicarea între cele două componente s-a făcut prin protocolul HTTP.

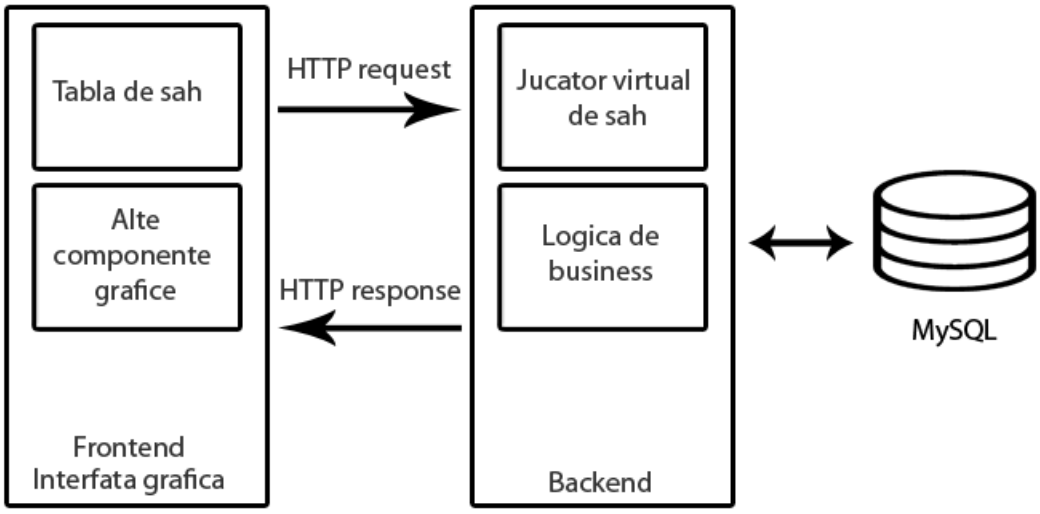

Figura 5.1 Arhitectura sistemului

În continuare este prezentată diagrama de componente a sistemului. Se pot distinge cele două componente mari, şi anume cea de frontent, interfață grafica și componența de backend.

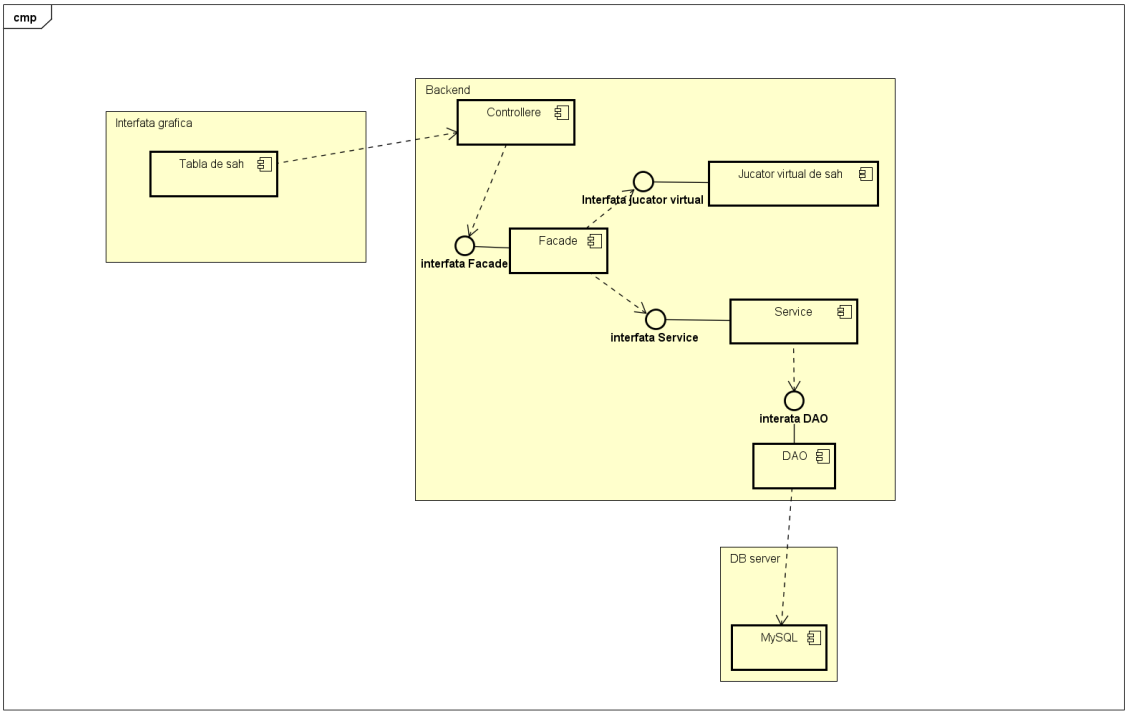

Figura 5.2 Diagrama de componente a sistemului

5.1.1. Arhitectura componentei de backend

Arhitectură serverului respectă modelul de arhitectură pe patru nivele, și anume: persistență, servicii, facade, controllere. Acesta este un model personalizat al arhitecturii three-tier.

Arhitectură mulți-tier este reprezentată de mai multe straturi, fiecare dintre acestea având un set de responsabilități bine definite. Aceste straturi sunt legate între ele printr-un lanț pentru a oferi funcționalitatea finală. Este important că aceste straturi să fie cât mai decuplate cu putință, pentru a crește gradul de reutilizare a acestora, ușurință de înlocuire, sau testarea acestora. Dacă aceste straturi sunt decuplate în mod corespunzător, se pot introduce și nivele de cache între ele pentru o performanță crescută.

Straturile folosite pentru backend-ul aplicației ChessEngine sunt după cum urmează:

- **Controller layer**: primește cererile de la utilizatori și apelează layerul de facade sau business pentru a procesa cererile. În final, un răspuns este returnat către utilizator.
- **Facade layer**: convertește informația din stratul de business în obiecte de tip DTO (Data Transfer Object) ce reprezintă vederi ale informației interne. Aceste obiecte au rolul de a ascunde detaliile ce nu sunt necesare în răspunsul ce va fi trimis către utilizator.
- **Business layer**: realizează logica de business a aplicației.
- **Persistence layer**: realizează legătura între stratul de business și baza de date.

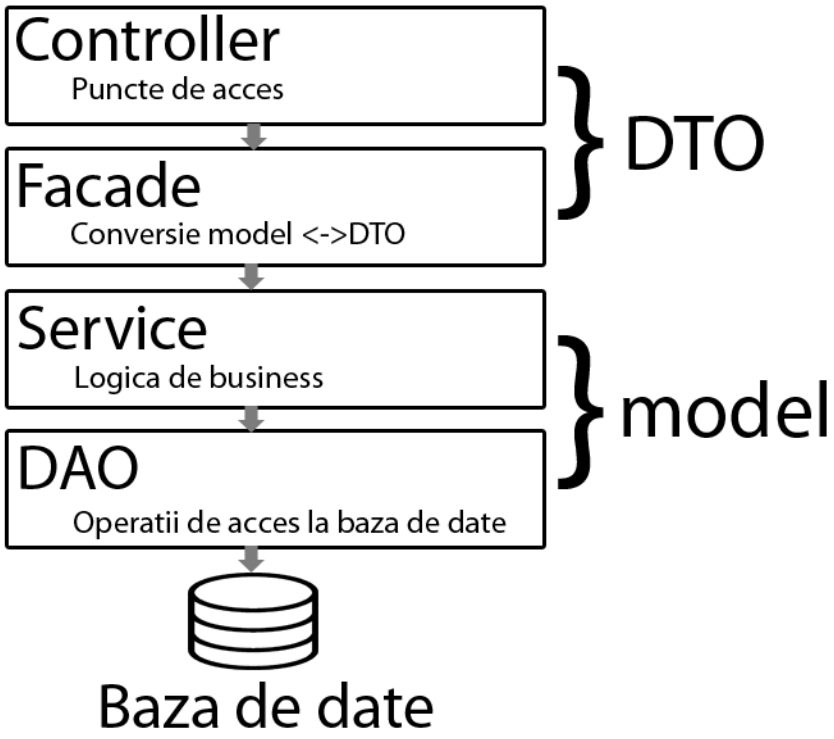

Figura 5.3 Arhitectura backend

5.1.1.1. Controller layer

Acest strat responsabil de interacțiunea utilizatorului cu sistemul. În cadrul acestui strat, sunt primite cererile pe care utilizatorul le plasează în interfață grafica și sunt returnate răspunsurile. Acest strat ascunde detaliile de implementare a sistemului prin apelarea stratului de business pentru a procesa cererea utilizatorului.

Acest strat este format din clase de tip Controller, ce sunt mapate la anumite url-uri și pot primi cereri de tip GET,POST, PUT sau DELETE, conform specificației protocolului HTTP.

5.1.1.2. Facade layer

Acest strat este responsabil cu conversia informației interne în obiecte de tip Data Transfer Object. Rolul acestor obiecte este de a oferi o vedere simplificată a informației interne. Acest strat este constituit din clase de tip Converter, ce convertesc un obiect de model (ce provine din stratul de business) într-un obiect de tip DTO (Data Transfer Object) și invers. Clasele de tip converter sunt folosite în cadrul claselor de tip Facade, ce apelează mai departe stratul de business pentru primirea informațiilor necesare răspunsului.

5.1.1.3. Business layer

Acest strat mai este numit și stratul de mijloc sau stratul de logică. Acest strat este responsabil cu logică de business a aplicației. Acest strat primește detaliile cererii și apelează stratul de persistență (Persistence layer). Informațiile ce vin din stratul de persistență sunt procesate în concordanță cu business-ul aplicației, iar răspunsul este trimis mai departe către presentation layer.

Stratul de business este alcătuit din clase de tip Service, unde este implementată logică aplicației. Exemple de astfel de clase sunt MessageService, clasă ce implementează logică pentru adăugarea, returnarea sau ștergerea mesajelor.

Tot în cadrul acestui strat este pus modelul aplicației. Clasele ce intră în model sunt necesare pentru reprezentarea informației. Astfel, există clase de model pentru utilizatori, pentru mesaje, pentru citate, pentru meciul de șah, pentru strategii sau probleme.

Între clasele de model există relații, spre exemplu o strategie are o succesiune de perechi de mutări: relație one-to-many, sau o pereche alb-negru de mutare are o mutare de alb și o mutare de negru (două relații one-to-one).

5.1.1.4. Modelul folosit

Pentru a reprezenta entitățile necesare jucătorului virtual de șah, a fost nevoie de modelare pentru utilizatori, meciurile de șah, strategiile, problemele de șah, noutățile, citatele și mesajele din aplicație.

Deoarece diagrama de model este mare, ea va fi spartă în bucăți. În continuare va fi prezentat modelul folosit pentru salvarea unui meci de șah:

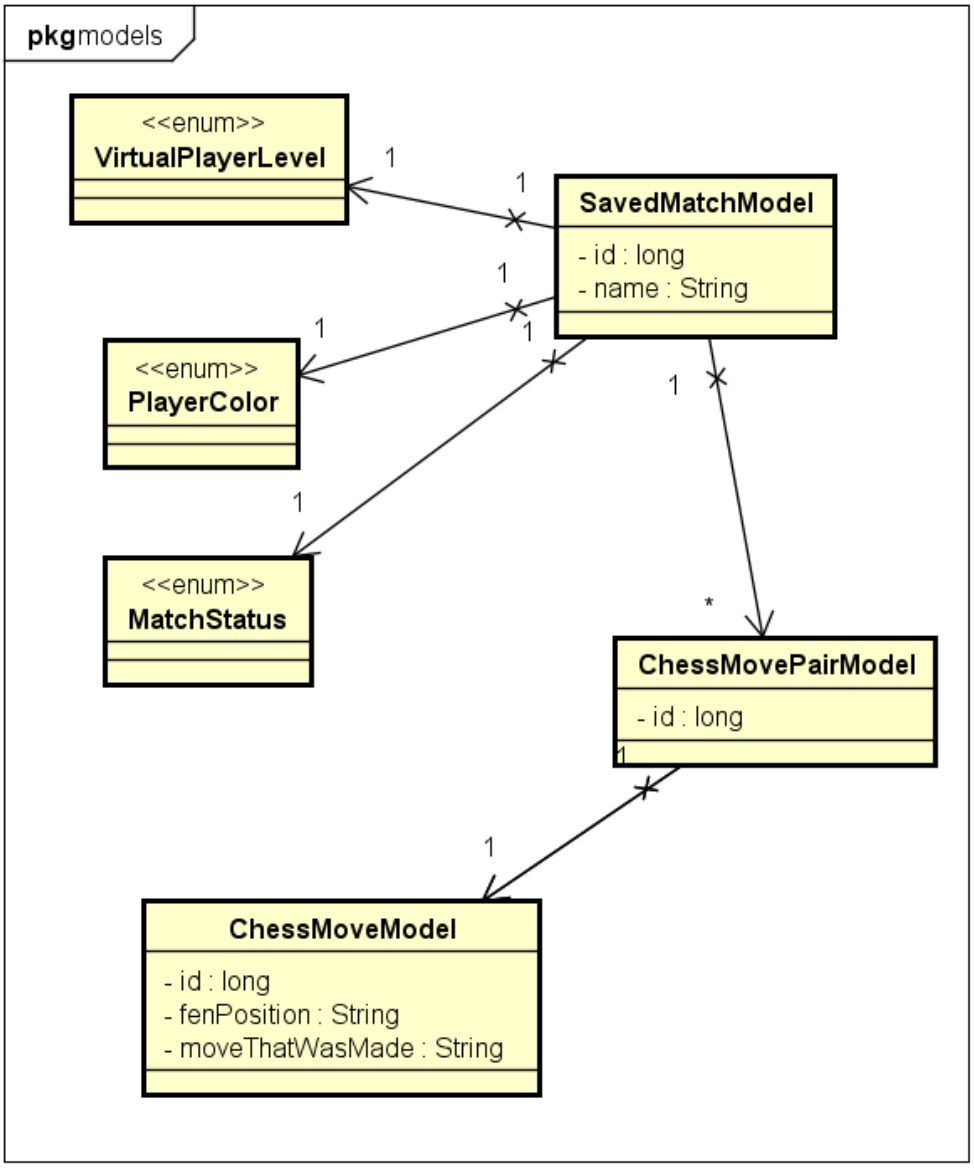

Figura 5.4 Diagrama UML model meci de șah

Pentru reprezentarea utilizatorilor, s-a folosit urmatorul model:

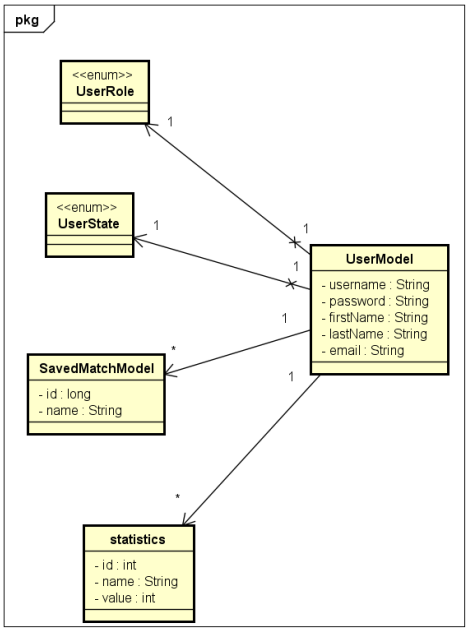

Figura 5.5 Diagrama UML model utilizator

Pentru a reprezenta o strategie de șah, s-a folosit următorul model:

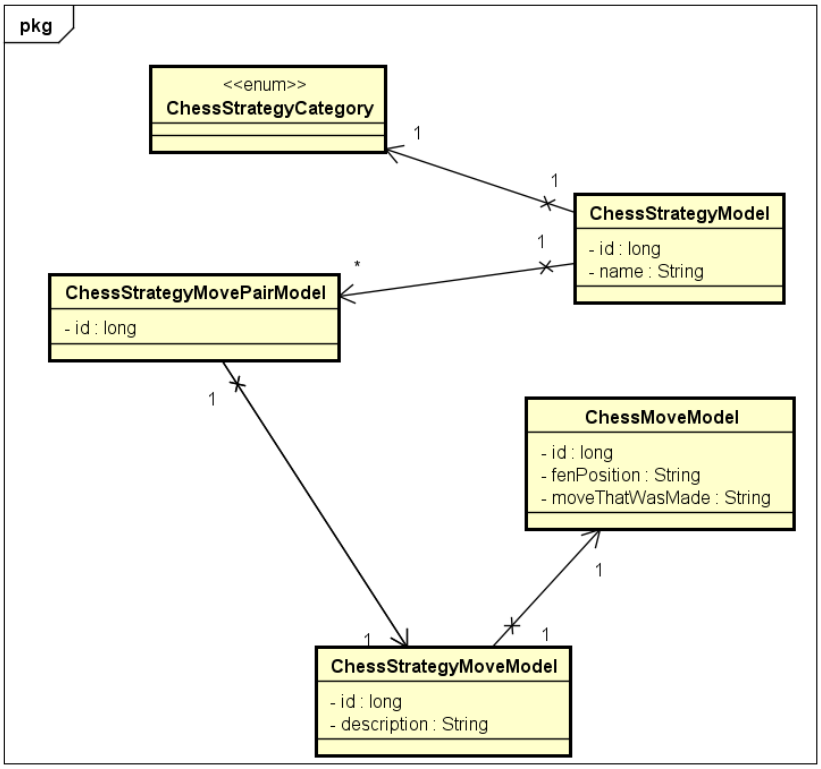

Figura 5.6 Diagrama UML model strategie de șah

5.1.1.5. Persistence layer

Acest strat este responsabil cu persistență datelor. În spatele acestul strat se află de obicei o bază de date. Asupra acestui strat se vor face doar operații simple, spre exemplu: citire, adăugare, ștergere, sau actualizare de date.

Acest strat este alcătuit din clase de tip DAO (data access object), ce conțin operații simple asupra bazei de date, cum ar fi adăugare, ștergere, citire sau actualizare. Pentru simplificarea proiectării, a fost creată o clasă de tip DAO generică ce conține operațiile: getAll(), getById(), update(), delete() și create(). Fiecare clasă din acest strat va extinde clasă generică, iar dacă va avea nevoie de operații suplimentare, atunci le va implementa. Principalul avantaj al acestei abordări este acela că dacă o clasă nu are nevoie de operații suplimentare, atunci singură responsabilitate a ei va fi să extindă clasă DAO generică.

Clasele de tip DAO apelează ORM-ul folosit pentru maparea claselor de model la tabelele din bază de date relațională: Hibernate, acest framework conectându-se la bază de date.

Pentru performanță operațiilor asupra bazei de date s-a folosit un connection pool. Acest concept presupune crearea unui set de conexiuni în momentul pornirii aplicației, denumit pool. Aceste conexiuni vor fi folosite de către operațiile asupra bazei de date. După ce o operație a fost terminată, conexiunea folosită va fi eliberată și va fi plasată înapoi în pool. Principalul avantaj al connection pool-ului este acela că odată creat, se evită operațiile de deschidere și închidere de conexiuni, operații ce au un timp semnificativ de execuție.

5.1.1.6. Proiectarea bazei de date

Baza de date a fost proiectată astfel încât să poată fi mapata la clasele de model din aplicație, astfel toate clasele din aplicație au câte o tabela echivalentă în bază de date și fiecare relație are o relație echivalentă în bază de date. Spre exemplu, relația dintre clasă strategie și clasă ce reprezeinta o pereche de mutare de one-to-many este prezența și între tabelele strategii și pereche_mutare din bază de date.

Pentru a demonstra că bază de date este în Formă Normală 3, în continuare se va face o trecere în revistă ale formelor normale inferioare.

Formă Normală 1 presupune faptul că nu există nicio coloană în bază de date pentru care să fie intrări cu mai multe valori. În cadrul bazei de date pentru aplicația ChessEngine nu există nicio tabela care să aibe astfel de atribute.

În Formă Normală 2, tabelele bazei de date trebuie să fie în Formă Normală 1 și să nu conțină dependințe parțiale între atributele acesteia și cheia primară. În cadrul bazei de date descrise în acest document nu există dependințe parțiale.

În Formă Normală 3, tabelele bazei de dat trebuie să fie în Formă Normală 2 și să nu conțină dependințe tranzitive între atributele acesteia și cheia primară, cu alte cuvinte, toate atributele non-prime să depindă doar de cheia primară. În cadrul bazei de date pentru aplicația ChessEngine nu există atribute non-prime care să nu depindă doar de cheia primară.

Constrângerile asupra tabelelor din bază de date au fost impuse cu ajutorul ORM-ului folosit, Hibernate, prin anotarea câmpurilor din clasele de model. Astfel există constrângeri referențiale ce presupun că dacă entitatea tare este actualizată sau ștearsă, această operație va fi făcută și asupra entităților slabe, referentiate de entitatea tare. De

asemenea există constrângeri de chei primare asupra identificatorilor fiecărei instanțe. Tipul acestor identificatori este „long”, iar valoarea lor este generată de către bază de date, pentru a evită coliziunile de identificatori.

Deoarece diagrama bazei de date are dimensiuni mari, această va fi spartă pe componente. Astfel, prima componența conține tabelele necesare persistarii unei probleme de șah:

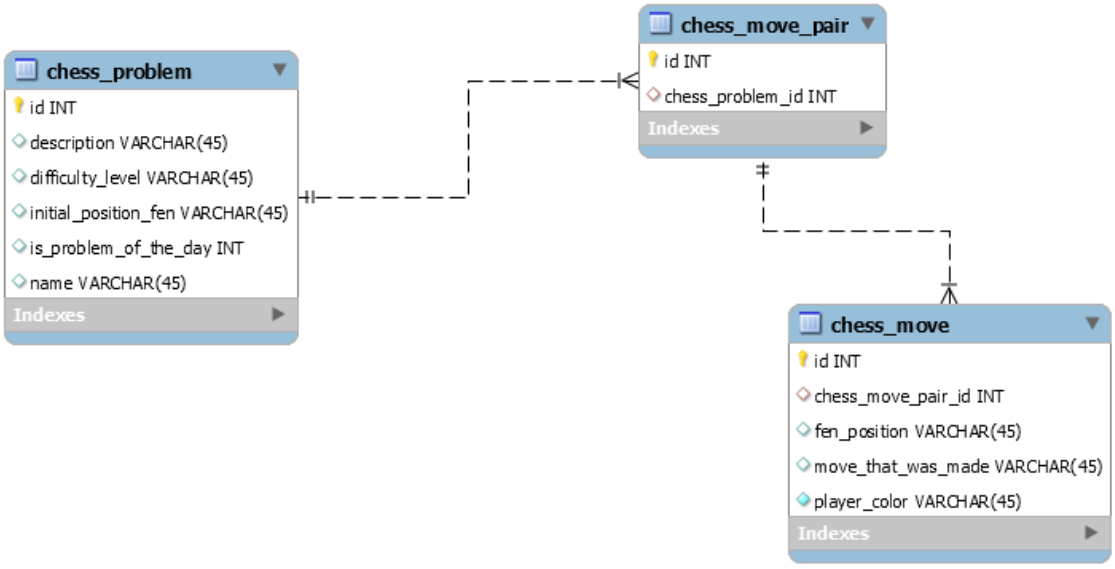

Figura 5.7 Diagrama bazei de date – probleme de șah

Pentru reprezentarea utilizatorilor, diagrama este ceva mai complexă, datorită faptului că pentru un utilizator sunt necesare salvările meciurilor pe care le-a jucat și statisticilor sale.

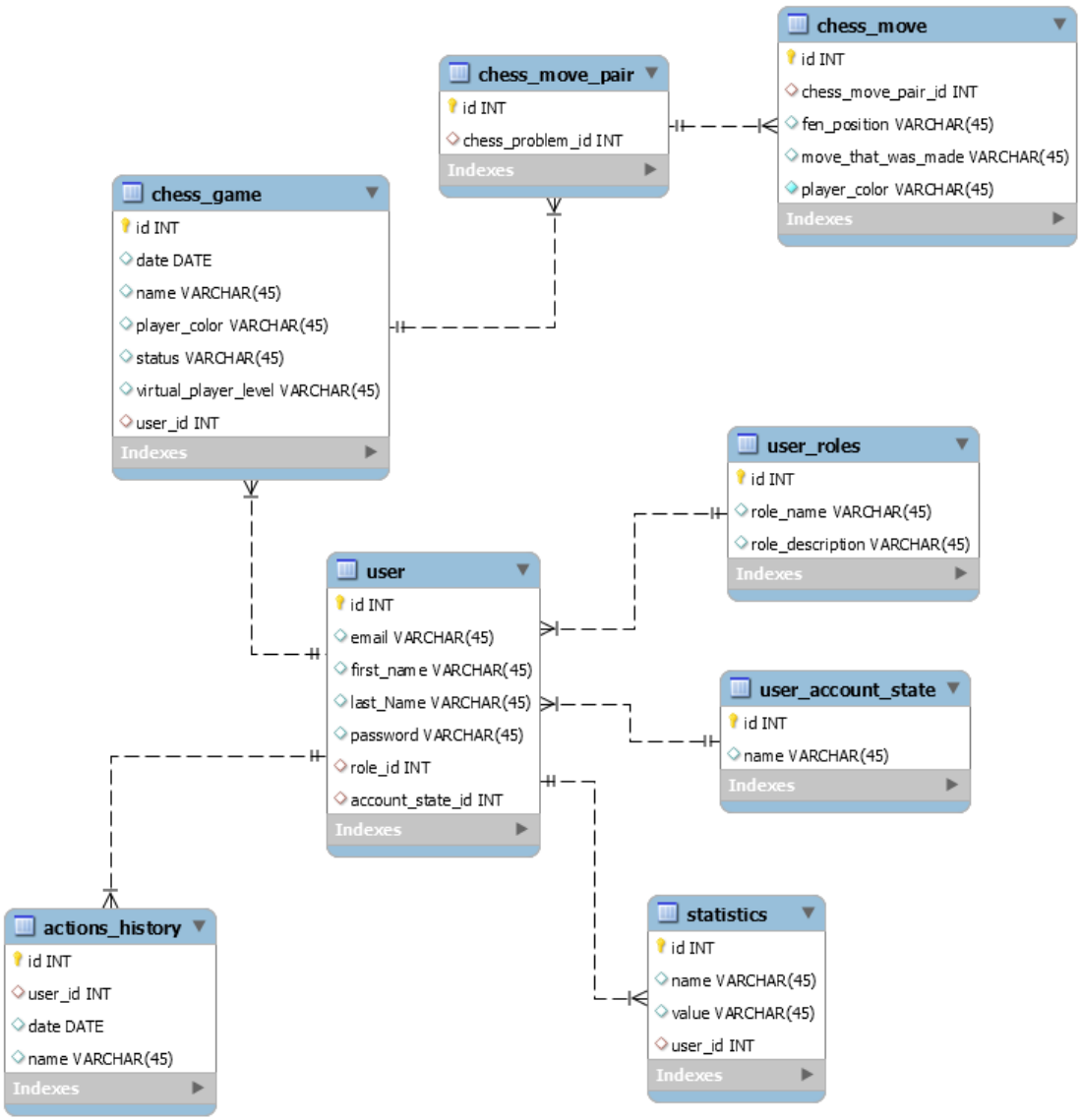

Figura 5.8 Diagrama bazei de date – probleme de șah

În continuare este prezentată diagrama necesară salvării unui meci de șah:

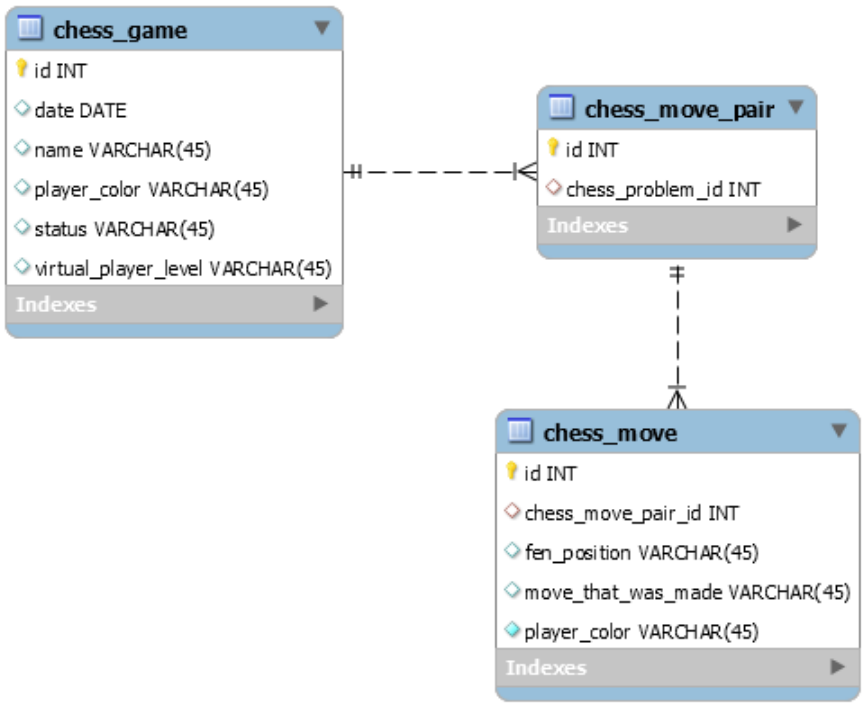

Figura 5.9 Diagrama bazei de date – probleme de șah

5.1.2. Arhitectura jucătorului virtual

Jucătorul virtual este componența centrală a aplicației ChessEngine, ea fiind localizată în partea de backend a aplicației.

Una dintre proprietățile ce le respectă această componența este aceea că jucătorul virtual nu are stare, prin urmare nu se rețin date de la o cerere la următoarea. Pentru fiecare cerere se specifică contextul (poziția pe tablă de șah, jucătorul ce trebuie să mute și nivelul de dificultate), acesta nefiind salvat. O altă proprietate a jucătorului virtual de șah este aceea că implementarea să este decuplată de restul aplicației, oferind posibilitatea că în viitor acesta să poată fi extras într-un serviciu web separat, ce să ruleze pe o arhitectură de tip cluster. O altă proprietate a jucătorului virtual de șah este aceea că fiecare componența are un grad minim de cuplare față de celalalte, pentru a ușura procesul de schimbare a uneia dintre ele. Spre exemplu, funcția de evaluare este abstractizată de o interfață, implementarea putând fi schimbată cu ușurință.

Jucătorul virtual primește că și parametrii de intrare poziția pe tablă de șah, culoarea jucătorului ce trebuie să mute (alb sau negru) și nivelul de dificultate la care trebuie să joace și generează că și ieșire mutarea următoare.

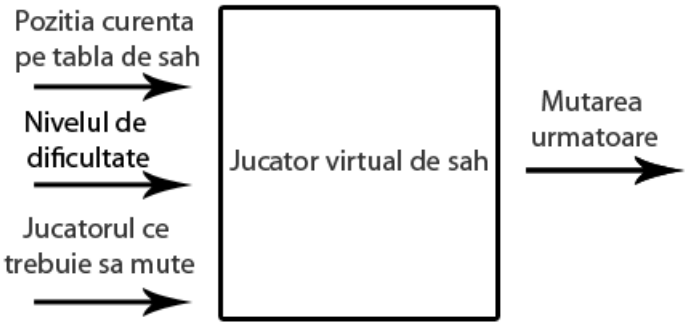

Figura 5.10 Intrări și ieșiri pentru jucătorul virtual de șah

În diagrama de componente ce urmează sunt prezentate principalele componente ale jucătorului virtual de șah, precum algoritmul central Minimax, funcția de evaluare, sau generatorul de mutări posibile:

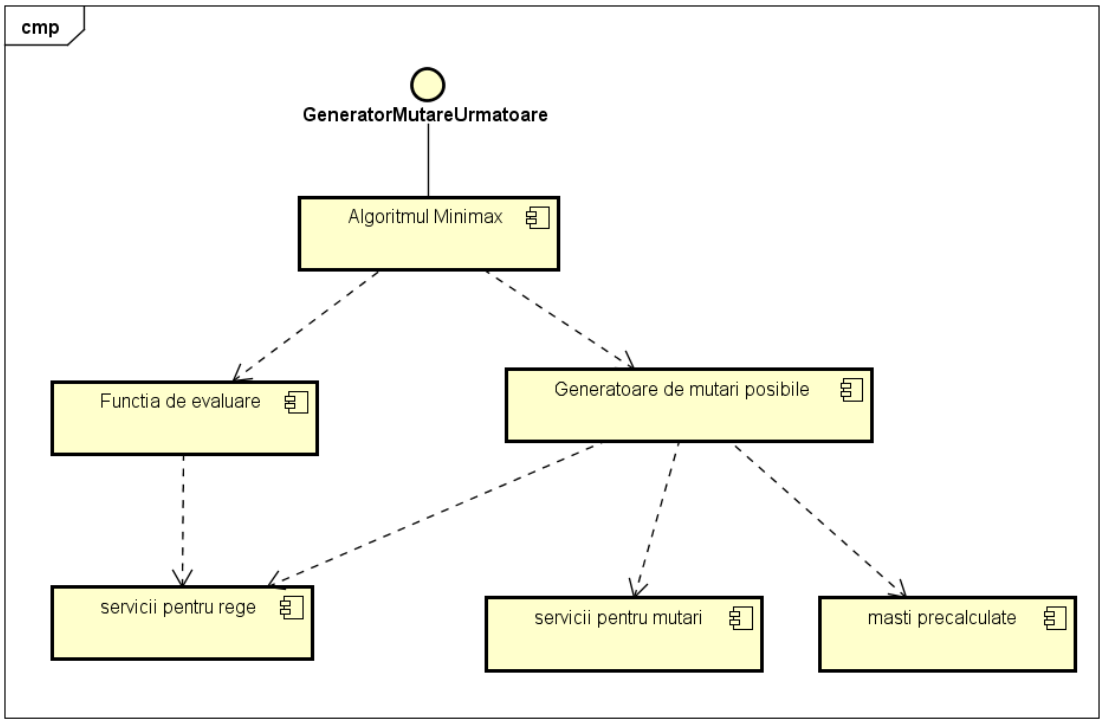

Figura 5.11 Diagrama de componente pentru jucătorul virtual de șah

În continuare vor fi prezentate aspecte legate de modul de reprezentare al datelor, algoritmul de căutare folosit, clasele ce generează mutări posibile și detecția sfârșitului meciului prin șah mat sau remiză.

5.1.2.1. Structuri de date folosite

Pentru a putea genera următoarea mutare, a fost nevoie de reprezentarea unui poziții de șah pe tablă și a unei mutări.

Pentru poziția pieselor de șah pe tablă este important de menționat faptul că trebuie să fie cât mai eficientă cu putință, deoarece în procesul de generare a mutării următoare se va genera un număr foarte mare de astfel de poziții. Spre exemplu, considerând că algoritmul va caută 2 mutări în avans, iar pentru fiecare poziție de șah există în medie 20 de mutări posibile, se vor genera aproximativ poziții de șah. De asemenea, este important că reprezentarea poziției de șah să ușureze procesul de generare a mutărilor posibile.

Datorită acestor constrângeri, s-a optat pentru bitboard-uri. Bitboard-urile sunt niște numere de tip long (pe 64 de biți) folosite în reprezentarea unei poziții pe tablă de șah. Se folosesc numere pe 64 de biți, deoarece o tablă de șah are 64 de câmpuri, iar fiecare bit din număr este mapat la un câmp de pe tablă de șah, astfel: bitul cel mai puțin semnificativ este mapat la câmpul A1 de pe tablă, iar bitul cel mai semnificativ este mapat la câmpul H8 de pe tablă. Fiecare bit de 1 din număr reprezintă o piesă pe tablă de șah, iar fiecare bit de 0 reprezintă un câmp gol. Pentru a reprezenta întreagă tablă de șah, se folosește câte un bitboard pentru fiecare tip de piesă, atât pentru alb, cât și pentru negru. Astfel, sunt necesare 12 bitboard-uri pentru reprezentarea unei poziții de șah pe tablă: 6 bitboard-uri pentru piesele albe și 6 bitboard-uri pentru piesele negre.

Această reprezentare a tablei de șah oferă în primul rând avantajul de a putea folosi operațiile pe biți pentru generarea mutărilor posibile, astfel crescând performanță aplicației. De asemenea, reprezentarea pe 64 de biți a bitboard-urilor se potrivește cu procesoarele pe 64 de biți, ceea ce oferă o creștere și mai mare a performanței.

Din punct de vedere al memoriei, bitboard-urile prezintă avantajul că sunt mai compacte, lucru ce permite o reducere semnificativă a memoriei utilizate pentru reprezentarea pozițiilor de șah.

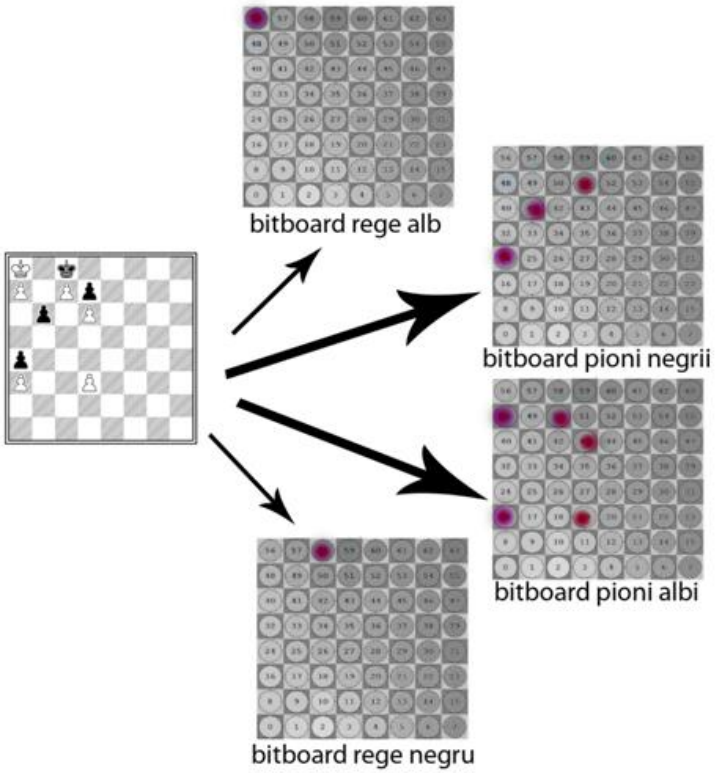

Figura 5.12 Arhitectura backend

Reprezentarea unei mutări de șah este de asemenea un pas ce poate influiență performanță aplicației sau memoria necesară acesteia. Pentru o reprezentare eficientă a mutărilor de șah, s-au folosit numere întregi, pe 32 de biți. O mutare este reprezentată printr-un astfel de număr pe 32 de biți.

În reprezentarea mutării trebuie sa apară următoarele informații:

- Poziția inițială – este câmpul în care se află piesă înainte de mutare. Acest câmp este reprezentat de coordonatele x și y de pe tablă de șah, x reprezentând numărul coloanei, iar y reprezentând numărul liniei. Deoarece există 8 linii și 8 coloane, este nevoie de 3 biți pentru reprezentarea numărului coloanei și 3 biți pentru reprezentarea numărului liniei, prin urmare este nevoie de 6 biți pentru reprezentarea poziției inițiale.
- Poziția finală – este câmpul în care va află piesă după mutare. Asemenea poziției inițiale, este reprezentat de coordonatele x și y pe tablă de șah, având nevoie de 6 biți pentru reprezentare.
- Piesa mutată – piesa ce este mutată în cadrul mutării. Fiind 12 piese, este nevoie de 4 biți pentru reprezentarea fiecărei piese.
- Rocada mică – este un „flag” ce este 1 dacă mutarea reprezintă rocada mică, sau 0 altfel.
- Rocada mare – este un „flag” ce are valoarea 1 dacă mutarea reprezintă rocada mare, 0 altfel.

- Este regele în șah - este un „flag” ce are valoarea 1 dacă mutarea pune regele advers în șah, 0 altfel.
- Piesa promovată – este piesă cu care a fost înlocuit pionul în cazul în care acesta a fost promovat la o altă piesă. Asemenea piesei mutate, este nevoie de 4 biți pentru a reprezenta piesă promovată.
- Piesa capturată – este piesă care va fi capturată în cadrul mutării. Asemenea piesei mutate, este nevoie de 4 biți pentru a reprezenta piesă capturată.
- Este șah mat - este un „flag” ce are valoarea 1 dacă mutarea pune regele advers în șah mat, 0 altfel.

De menționat este faptul că informațiile cele mai importante apăr în partea cea mai semnificativă a numărului pe 32 de biți, iar restul apăr în partra mai puțin semnificativă. Astfel flag-ul de șah mat este pus e poziția 27, piesă capturată ocupă pozițiile de la 23 la 26 și așa mai departe. Astfel, o valoare mai mare a numărului indică o mutare mai bună pentru jucătorul ce face mutarea, iar o valoare mai mică, o mutare mai slabă din partea jucătorului. Astfel prin compararea valorilor a numere pe 32 de biți ce reprezintă mutări, este echivalentă cu evaluarea calității mutărilor.

Figura 5.13 Reprezentarea mutării de șah

5.1.2.2. Funcția de evaluare

Funcția de evaluare are rolul de a oferi un scor pentru o anumită poziție pe tablă de șah. Algoritmul Minimax folosește această funcție pentru a evalua pozițiile finale, cu scopul de a o alege pe cea mai bună.

Funcția de evaluare pe care am folosit-o se bazează într-o oarecare măsură pe avantajul material, dar și pe alte aspecte, cum ar fi siguranță regelui sau pozițiile celorlalte piese.

Pentru calculul avantajului material, s-a folosit diferență între numărul de piese din acel tip pentru alb și piesele echivalente negre. Fiecare tip de piesă are asociată o

pondere, deoarece regină spre exemplu, este mai valoaroasa decât un pion. Astfel, regină are ponderea 300, turnul 150, nebunul 100, calul 90 iar pionul 70.

Siguranță regelui este reprezentată de numărul de câmpuri în care regele se poate mută în mod legal. Evaluarea poziției anumitor piese cum ar fi pionii, sau caii s-a făcut prin utilizarea unor măști precalculate. Măștile sunt reprezentate de tablouri cu 64 de valori, fiecare valoare fiind mapata la un câmp de pe tablă de șah.

5.1.2.3. Implementarea claselor care generează mutări posibile

Clasele care generează mutări posibile vor fi apelate de foarte multe ori în decursul procesului de găsire a mutării următoare, de aceea este important că acestea să opereze într-un mod eficient, evitând calculele inutile. Pentru generarea mutărilor următoare s-au folosit atât operații pe biți, cât și măști precalculate, obținând o performanță ridicată.

Pentru o structurare corectă a codului, s-a folosit mecanismul de polimorfism, astfel există o interfață generală PossibleMovesGenerator ce este implementată de fiecare generator de mutări posibile.

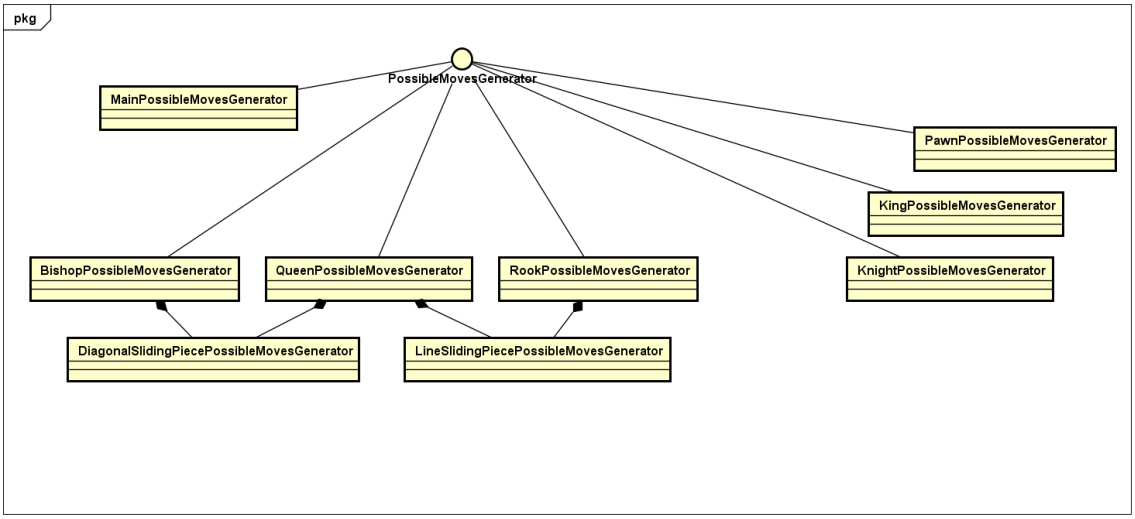

Figura 5.14 Diagrama UML pentru clasele ce generează mutări posibile

Cele două tehnici vor fi descrise în continuare, în corelație cu prezentarea generării mutărilor posibile pentru fiecare tip de piesă.

Pentru o mai ușoară înțelegere, vor fi descrise în primul rând generatoarele de mutări posibile ce folosesc operații pe biți.

Pentru generarea mutărilor pionilor de alb, s-au folosit următoarele operații pe biți asupra bitboard-ului de pioni: bitboard shiftat cu 8 poziții spre stânga, pentru a obține mutările cu o poziție în față ale tuturor pionilor de alb, bitboard shiftat cu 16 poziții spre stânga, pentru a obține mutările cu două poziții în față ale tuturor pionilor de alb, bitboard shiftat cu 7 poziții spre stânga, pentru a obține capturările la stânga pentru toți pionii de alb și bitboard shiftat cu 9 poziții spre stânga, pentru a obține capturările la dreaptă pentru toți pionii de alb.

Pentru generarea mutărilor posibile pentru cal, s-au folosit operații pe biți pentru a genera fiecare dintre cele 8 mutări posibile pentru un cal. Astfel, spre exemplu pentru a mută calul în poziția sud-est-est, s-a shiftat bitboard-ul corespunzător cailor cu 6 poziții spre dreaptă, iar pentru a mută calul în poziția nord-nord-est, s-a shiftat bitboard-ul corespunzător cailor cu 17 poziții spre stânga.

În continuare vor fi prezentate generatoarele de mutări posibile ce folosesc măști precalculate. Aceste măști precalculate vin în sprijinul sistemului, aducând un plus de performanță prin evitarea recalcularii pozițiilor pentru un anumit număr de ori. Pentru fiecare tip de piesă pentru care mutările posibile se generează utilizând măști precalculate, există o lista cu 64 de numere pe 64 de biți, fiecare dintre aceste numere reprezentând un bitboard cu pozițiile pe care o piesă poate să mute în cazul în care este singură pe tablă.

În cazul generatorului de mutări pentru rege, se ia masca ce corespunde poziției regelui și se scot pozițiile în care regele nu poate fi mutat, datorită ocupării acelor poziții de către alte piese, sau datorită faptului că în acele câmpuri regele ar fi în șah. Formulă folosită pentru generarea mutărilor posibile pentru rege este după cum urmează:

Bitboard_mutări_posibile_rege = masca_precomputata & ~bitboard_cu_toate_piesele & ~masca_precalculata_rege_aversar & ~bitboard_cu_câmpurile_atacate_de_adversar;

De menționat este faptul că anumite operații pe biți au fost în continuare necesare, dar prin folosirea măștilor precalculate, au fost evitate un număr considerabil de operații.

Generatorul de mutări legale pentru turn și pentru nebun au fost refolosite pentru generarea mutărilor reginei, deoarece regină toate face toate mutările pe care le pot face nebunul și turnul la un loc.

5.1.2.4. Detectarea șah-matului

Responsabilitatea de detectare a șah-matului a fost asignata serviciului KingService, ce conține o metodă getKingStateAfterMove(). Această metodă primește că parametrii de intrare poziția curentă pe tablă de șah și mutarea ce va fi făcută. Această metodă poate returna două valori posibile: CHECK_MÂȚE sau KING_ÎN_CHECK.

Algoritmul folosit este următorul:

```
daca(este_regele_in_șah(pozitia_de_șah_dupa_mutare)) atunci
        mutari_posibile=genereaza_mutarile_posibile();
        daca(nu exista mutari_posibile) atunci
                returneaza CHECK_MATE
        sfarsit daca

        returneaza KING_IN_CHECK
sfarsit daca
```

5.1.2.5. Secvența pentru generarea mutării următoare

În figură următoare este prezentat un flow complet pentru generarea mutării următoare. Se observă trecerea prin straturile aplicației, și anume Controller, Facade și jucătorul virtual de șah.

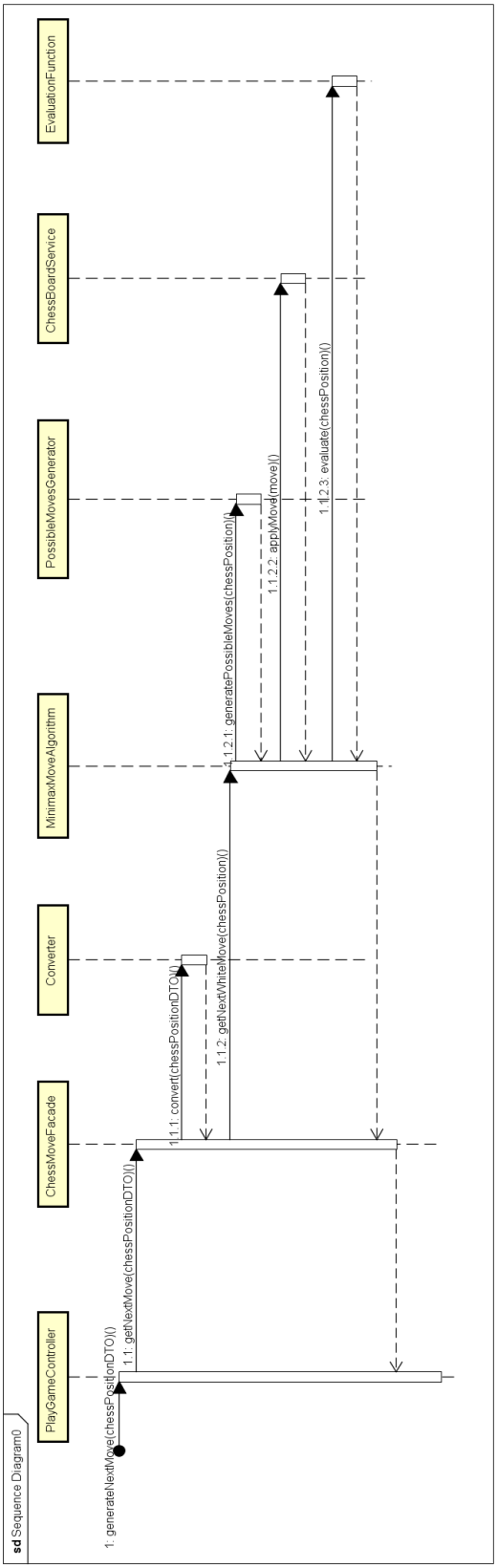

Figura 5.15 Diagrama de secvența pentru generarea mutării următoare

5.1.3. Arhitectura componentei de frontend

Pentru componența de frontend s-a folosit limbajul jsp pentru adăugarea conținutului dinamic în paginile statice. Cu ajutorul acestui limbaj s-au realizat șabloane pe bază componentelor grafice ce se repetă în mai multe pagini, spre exemplu, secțiunea de header sau footer.

5.1.4. Logging-ul in aplicatie

Logging-ul a fost introdus în aplicație cu scopul de a ușura descoperirea eventualelor probleme apărute. Principalul avantaj al folosirii unui astfel de framrwork este acela că mesajele pot fi blocate, sau pornite în timp ce aplicația rulează, fără a mai fi nevoie de build-urea sistemului.

Framework-ul de logging pentru care s-a optat este log4j, s-a optat pentru acest framework, deoarece oferă doar funcționalitatea de bază și nu are implicații negative semnificative asupra performanței aplicației.

Pentru adăugarea mesajelor de logging în aplicație, s-a adoptat următorul set de convenții:

- În fiecare mesaj de logging, s-a adăugat și contextul în care a apărut evenimentul ce este logat.
- Pentru toate acțiunile utilizatorului, s-au plasat mesaje de logging în controller, având nivelul INFO.
- Pentru fiecare excepție aruncată, se va loga un mesaj cu nivelul ERROR, fiind inclusă și cauza excepției în mesaj.
- Se vor evita situațiile de NullPointerException în cadrul mesajelor, deoarece acestea pot provoca mai multe confuzii.

5.1.5. Design patterns folosite în aplicație

Pentru a crește gradul de înțelegere a codului sursă, dar și a ușura procesul de mentenanța, au fost folosite design pattern-uri în aplicație.

5.1.5.1.Adapter

Există situații în care este necesară comunicarea între două componente ce au interfețe diferite, fiind necesar un mecanism de conversie a obiectelor transferate între aceste componente.

Design pattern-ul Adapter intervine în rezolvarea acestei probleme prin realizarea unui pod între cele două componente, reprezentat sub formă de convertor[8].

În aplicația ChessEngine, acest model a fost folosit pentru convertirea poziției pe tablă de șah din reprezentarea FEN în reprezentarea internă a jucătorului virtual de șah. Această conversie este realizată în stratul Facade al aplicației.

Pentru o reprezentare vizuală a acestui model de proiectare, este prezentată o diagrama UML în cadrul secțiunii Facade pattern de mai jos.

5.1.5.2. Facade pattern

Pentru a decupla componentele și a crește gradul de reutilizare și mentenanța în cadrul unui sistem, este necesară izolarea complexității unei componente. Design pattern-ul Facade ascunde complexitatile unei componente față de client, prin expunerea unor metode ce returnează obiecte ce conțin doar informația necesară clientului, simplificând astfel comunicarea cu acesta.

În cadrul aplicației, acest design pattern a fost folosit pentru a ascunde detaliile de implementare în componența de backend, fiind reprezentat de un strat denumit Facade. Spre exemplu, în cadrul procesului de generare a următoarei mutări, nu este necesară expunerea tuturor detaliilor din reprezentarea internă a mutării, cum ar fi piesă capturată, sau flag-ul ce reprezintă șah mat, ci doar poziția inițială și cea finală a mutării. Astfel, stratul facade oferă stratului Controller doar această informație.

Aceast model este reprezentat în diagrama de mai jos, în care clasa Controller este client pentru clasa Facade:

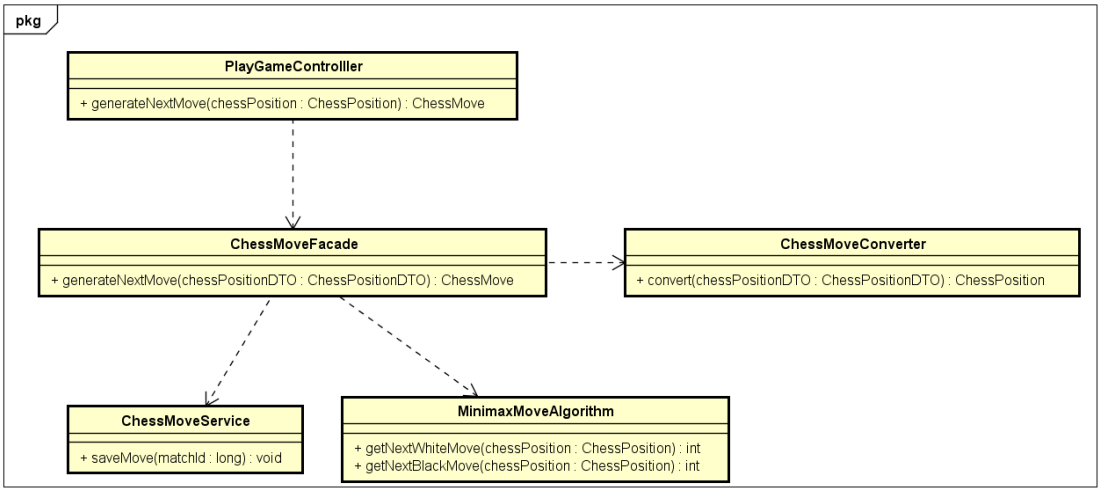

Figura 5.16 Diagrama de clase pentru modelul de proiectare Facade

5.1.5.3. Strategy

Există situații în care clasele diferă doar prin comportamentul lor. Pentru aceste situații este recomandat să se realizeze o interfață comună ce să fie apoi implementată de toate clasele ce diferă doar prin comportament[8].

În cadrul jucătorului virtual, acest design pattern a fost folosit pentru generarea mutărilor posibile. Fiecare generator de mutări posibile primește că și parametru de intrare o poziție pe tablă de șah și returnează o lista de mutări posibile, astfel a fost realizată o interfață ce abstractizează această situație, interfață ce a fost implementată de fiecare generator de mutări posibile în parte.

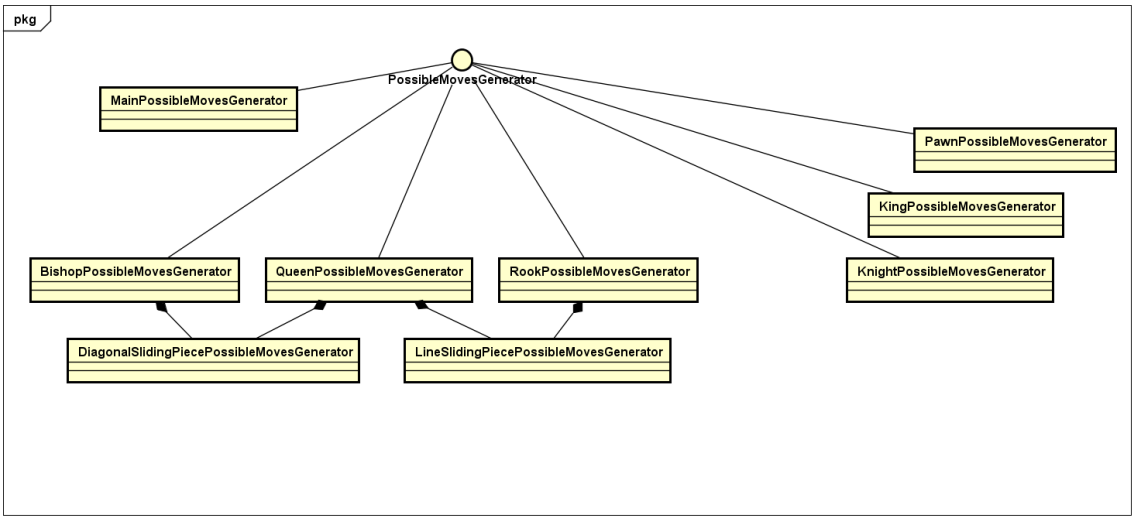

Figura 5.17 Diagrama UML pentru clasele ce generează mutări posibile folosind Strategy ca și model de proiectare

5.1.5.4. Data access object

Există situații în care este necesară comunicarea cu un sistem extern ce are rol de persistare a datelor, cum ar fi o bază de date spre exemplu. Pentru a izola complexitatea bazei de date și a decuplă tipul bazei de date folosit de aplicație, se folosește modelul Data Access Object pentru a realiză o interfață ce abstractizează bază de date. Acest model realizează are în componența doar operații simple, cum ar fi citire, adăugare, actualizare, sau ștergere.

În cadrul aplicației ChessEngine, acest design pattern a fost folosit pentru accesul la bază de date, fiind reprezentat de un strat denumit DAO în cadrul componenței de backend.

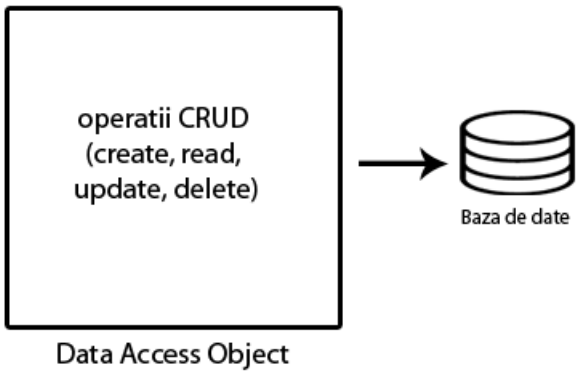

Figura 5.18 Reprezentarea modelului de proiectare Data Access Object

5.1.5.5. Object pool

Există anumite obiecte în Java care necesită mai multe resurse pentru a fi create. Un exemplu de astfel de obiect este conexiunea la bază de date.

Pentru a evită fabricarea excesiva a conexiunilor la bază de date, s-a folosit un așa-numit „pool" de conexiuni, ce funcționează după cum urmează: la momentul pornirii aplicației este creat un număr configurabil de conexiuni la bază de date, acestea fiind puse într-un „pool". Astfel, de fiecare data când este nevoie de o conexiune la bază de date este luat un obiect de tip conexiune din „pool", este folosit, apoi este pus înapoi în „pool" [8]. Datorită acestui design pattern, performanță aplicației crește, prin scăderea timpului de răspuns, datorat faptului că nu mai este necesară crearea unui obiect de conexiune la bază de date pentru fiecare cerere în parte.

5.1.6. Metodologia de dezvoltare a aplicatiei

Datorită dimensiunilor remarcabile ale acestui proiect, este nevoie de o metodologie de dezvoltare a software-ului pentru a fi dus la bun sfârșit. Metodologia folosită în realizarea aplicației ChessEngine este Unified Process. Astfel, dezvoltarea aplicației a fost împărțită în mai multe faze descrise în paragrafele următoare.

Fază incipientă a fost compusă din studiul bibliografic, analiză sistemelor existențe și construirea unui set primar de cerințe funcționale și non-funcționale. Pentru a organiza aceste informații, a fost elaborat documentul de viziune. Tot în cadrul acestei faze a fost dezvoltat un plan de lucru, în care fiecare iterație avea o data propusă pentru finalizare. În cadrul acestei etape au fost mai multe iterații, fiecare dintre acestea terminându-se prin evaluarea făcută de către coordonatorul de lucrare. În iterația următoare se mai adăugau intrări în documentele amintite mai sus și se reparau eventualele probleme semnalate prin feedback.

Fază de elaborare s-a focusat în mare parte pe definirea cazurilor de utilizare, definirea actorilor sistemului. De asemenea a fost planificată arhitectură sistemului și s-au stabilit tehnologiile ce au fost folosite în fază următoare. În cadrul acestei faze a fost dezvoltat un prototip al jucătorului virtual ce știa să mute doar pionii și au fost făcute țeste de performanță pentru a determină timpul necesar generării de mutări. Cazurile de testare au fost compuse din generarea în mod repetat a 10 000 de mutări de pioni, timpul de execuție fiind sub 3 secunde. Tot în cadrul acestei faze a fost proiectată interfață grafica, această fiind la sfârșitul fazei de elaborare din pagini statice. În cadrul acestei faze au existat două iterații, fiecare dintre ele sfârșindu-se cu evaluarea făcută de coordonatorul științific.

Fază de construcție a fost alcătuită din mai multe iterații ce au inclus dezvoltarea jucătorului virtual și a aplicației web în paralel. În prima iterație a fost dezvoltat scheletul aplicației și a fost făcut setup-ul aplicației, proces ce a inclus instalarea de framework-uri necesare, medii de dezvoltare și adăugarea sistemului de versionare ales, și anume GÎT [20]. În iterațiile următoare au fost dezvoltate pe rând funcționalitățile sistemului. Fiecare funcționalitate era implementată în mod vertical, ceea ce însemna că pentru fiecare caz de utilizare erau implementate pe rând clasele din stratul de persistență, serviciile, facadele și controllerele necesare, apoi era integrată componența din interfață grafica necesară cazului de utilizare sistem.

La sfârșitul acestei faze, jucătorul virtual de șah fost integrat aplicație.

de tranziție fost din efectuate aplicației. Spre exemplu, pentru testarea de „cutie neagra”-analizat -au testat cazurile de utilizare. Pentru testarea jucătorului virtual au fost jucate meciuri de șah. Defectele detectate au fost reparate cadrul acestei faze.

5.1.6.1. Versionarea datelor folosind GIT

Pentru a asigură integritatea sistemului și a ușura dezvoltarea acestuia, precum și integrarea continuă, s-a recurs la folosirea unui sistem de versionare: GIT. Principalele avantaje ale acestei abordări sunt după cum urmează: posibilitatea dezvoltării aplicației pe mai multe ramuri, copii de rezervă realizate în permanență, posibilitatea dezvoltării aplicației de pe mai multe stații și vizualizarea progresului prin analizarea versiunilor sau reducerea riscului de ștergere sau modificare accidentală a unor fișiere din proiect. Au fost folosite în total două repository-uri de GIT: unul pentru codul sursă al aplicației, iar celălalt pentru documentația proiectului. Deoarece în realizarea proiectului a fost implicată o singură persoană, s-a folosit un singur branch pentru fiecare repository de GIT.

Capitolul 6. Testare şi Validare

În acest capitol se dorește prezentarea principalelor modalități de testare a sistemului ce s-au folosit pe parcursul dezvoltării aplicației. Rolul acestor țeste este de a valida cerințele funcționale și nonfunctionale impuse.

Pe parcursul dezvoltării aplicației, au fost folosite următoarele mecanisme de testare: testare unitară, folosind framework-ul Junit, testare de tip cutie neagră. Pentru evaluarea aplicației au fost implicați și utilizatorii, prin folosirea unui chestionar pe care aceștia l-au completat.

6.1. Testare unitară

Datorită faptului că metodologia a impus folosirea unui proces iterativ de dezvoltare, la finalul fiecărei iterații a fost nevoie de testare pentru funcționalitățile implementate. Astfel, spre exemplu la sfârșitul iterației în care a fost implementat jucătorul virtual de șah, au fost scrise țeste unitare asupra claselor de generare a mutărilor posibile.

Testarea s-a realizat manual, fiind scrise țeste unitare în special pentru partea de jucător virtual, unde erorile de implementare pot duce la o funcționare defectuasa a sistemului. De asemenea, au fost testate funcționalitățile sistemului specificate în cadrul cerințelor funcționale.

Framework-ul folosit pentru testarea unitară este Junit, acesta fiind principalul mecanism de testare folosit în aplicațiile scrise în limbajul de programare Java.

6.2. Testarea performanței

Testele de performanță implică testarea tîmpilor de răspuns și modul în care reacționează sistemul la heavy load. Aceste țeste sunt necesare pentru a demonstra faptul că aplicația îndeplinește cerințele non-funtionale ce vizează performanță aplicației.

Pentru a realiză testele de performanță, a fost folosit JMeter, un tool de tip open source folosit pentru testarea performanței aplicațiilor web. JMeter este o aplicație de tip desktop ce pune la dispoziție o interfață grafica, având următoarele funcționalități: testare orientată pe performanță asupra funcționalităților de business, țeste de tip regression și load testing. Aceste țeste trebuie să demonstreze faptul că software-ul îndeplinește cerințele non-funcționale ce se referă la performanță aplicației.

Aplicația web a fost instalată pe un calculator personal, având un procesor Intel Core i7 4710MQ cu 4 nuclee și memorie RAM de 8 GB.

Pentru testare a fost folosite mai multe planuri de test, planuri ce sunt reprezentate în tabelul de mai jos:

Tabel 6.1 Testare de performanță pagini fără jucătorul virtual de șah

Test plan id	Nr. Utilizatori concurenți	Nr cereri/ utilizator	Interval între 2 cereri de la același utilizator	Timp minim de răspuns (ms)	Timp maxim de răspuns (ms)	Timp mediu de răspuns (ms)	Nr. Cereri procesate/ secundă
1	100	8	2	15	1099	335	190,9
2	100	8	3	13	892	246	188,5
3	100	10	3	16	1345	280	191.3
4	100	11	3	15	1715	237	209,3
5	100	12	3	16	1317	282	205,1
6	100	13	3	11	1295	284	228,9

În tabelul de mai sus sunt prezentate rezultatele testelor de performanță executate asupra paginii principale a aplicației, pagină ce nu implică activitatea jucătorului virtual de șah. Se constată faptul că timpul mediu de răspuns este în jurul valorii de 290 de milisecunde la un număr de 100 de utilizatori concurenți, iar timpul maxim depășește 1 secundă în cazurile de heavy load, cum ar fi în situația în care sunt 100 de utilizatori concurenți ce termini câte 10,11,12 respectiv 13 cereri la interval de 3 secunde.

În tabelul următor sunt prezentate rezultatele testelor de performanță pentru o pagină ce implică jucătorul virtual de șah, și anume generarea mutării următoare. Pentru aceste țeste se va folosi o poziție de început pe tablă de șah, jucătorul virtual operând la nivelul 4 de dificultate, acesta fiind nivelul maxim admis.

Tabel 6.2 Testare de performanță pagini cu jucătorul virtual de șah

Test plan id	Nr. Utilizatori concurenți	Nr cereri/ utilizator	Interval între 2 cereri de la același utilizator	Timp minim de răspuns (ms)	Timp maxim de răspuns (ms)	Timp mediu de răspuns (ms)	Nr. Cereri procesate/ secundă
1	10	2	1	5092	10015	7380	52,6
2	9	2	1	5073	10236	7075	51,3
3	8	2	1	4277	9586	6375	48,9
4	7	2	1	4505	9534	6378	42,6
5	6	2	1	3407	9997	6105	41,3
6	5	2	1	4487	9833	6835	28,9

Din tabelul de mai sus se poate observă că timpul mediu de răspuns este undeva între 6 și 7 secunde, cu un număr variabil între 5 și 10 utilizator concurenți. Timpul

maxim de răspuns a depășit 10 secunde numai în situațiile cu 10 respectiv 9 utilizatori concurenți.

6.3. Chestionar adresat utilizatorilor

Pentru a testa gradul de utilizabilitate al aplicației, s-a folosit un chestionar cu un set de întrebări legate de interfață grafica. Chestionarul a fost publicat cu ajutorul Google Forms și distribuit la utilizatorii aplicației. În urmă evaluării, au fost obținute următoarele rezultate:

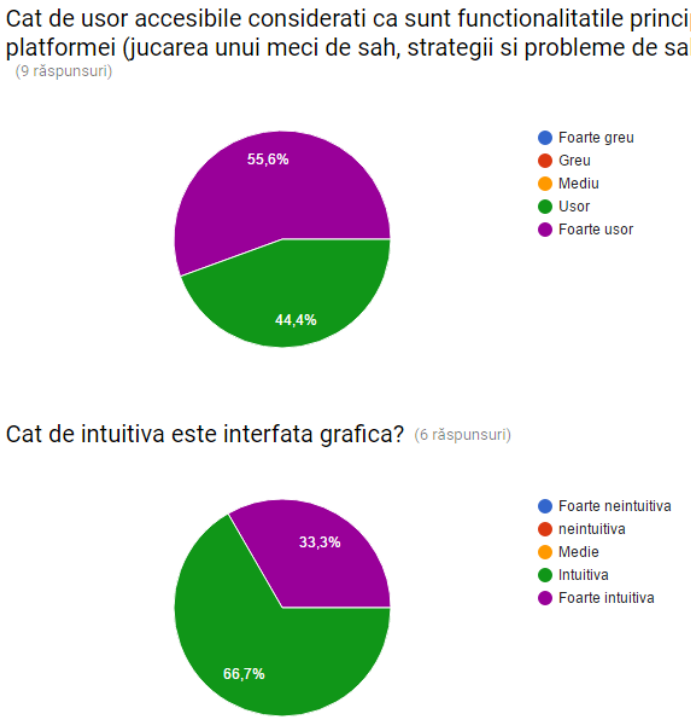

Figura 6.1 Rezultate chestionar aplicație

După cum se poate observă, aproximativ 55% dintre utilizatori consideră că funcționalitățile principale ale aplicației sunt ușor accesibile, iar interfață grafica este suficient de intuitivă, primind un rezultat de 66% pentru foarte intuitivă și 33% pentru intuitivă. Scorul per total obținut prin evaluarea interfeței grafice a fost 4,12 din 5.

Ca și evaluare a calității mutărilor jucătorului virtual de șah, a fost obținut un scor de 4,57 din 5. În ceea ce privește timpul de răspuns al jucătorului virtual de șah, 60% din utilizatori s-au declarat mulțumiți, restul fiind foarte mulțumiți.

Capitolul 7. Manual de Instalare și Utilizare

În continuare va fi prezentat manualul de instalare și utilizare al aplicației.

7.1. Instalarea și rularea

Pentru instalarea aplicației, este nevoie de următorul set de tool-uri: JRE, Tomcat, MySQL. Există două variante pentru instalarea aplicației. Prima variantă presupune faptul că utilizatorul are fișierul .war în posesie.

În acest caz, el va plasă fișierul cu extensia .war în următorul folder din Tomcat: apache-tomcat-8.0.33webapps, că în imaginea de mai jos:

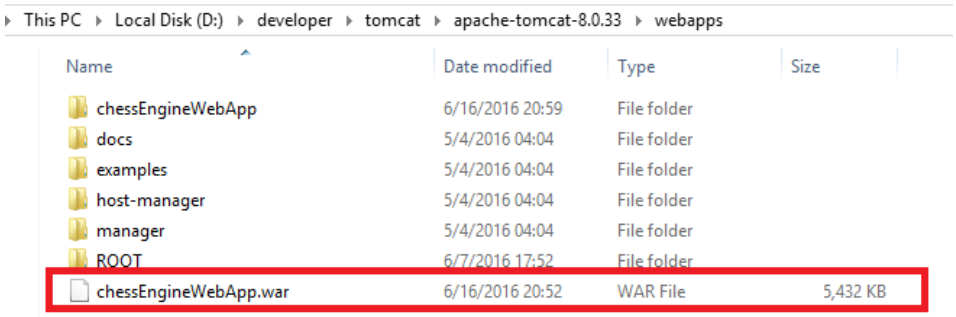

Figura 7.1 Deployment-ul aplicației în Tomcat

Se va porni server-ul de MySQL de prima data.

Se va porni serverul de Tomcat astfel: se va naviga la următorul folder:

apache-tomcat-8.0.33bin și se va rulă fișierul batch denumit startup.băț. Serverul de Tomcat va porni și va face deployment-ul aplicației. În final, aplicația va fi accesibilă folosind următorul url:

http://localhost:9090/chessenginewebapp

Acest url va fi deschis într-un browser la alegere. Dacă deployment-ul aplicației a fost realizat cu succes, atunci în browser ar trebui să apăra interfață grafica a aplicației, că în imaginea de mai jos:

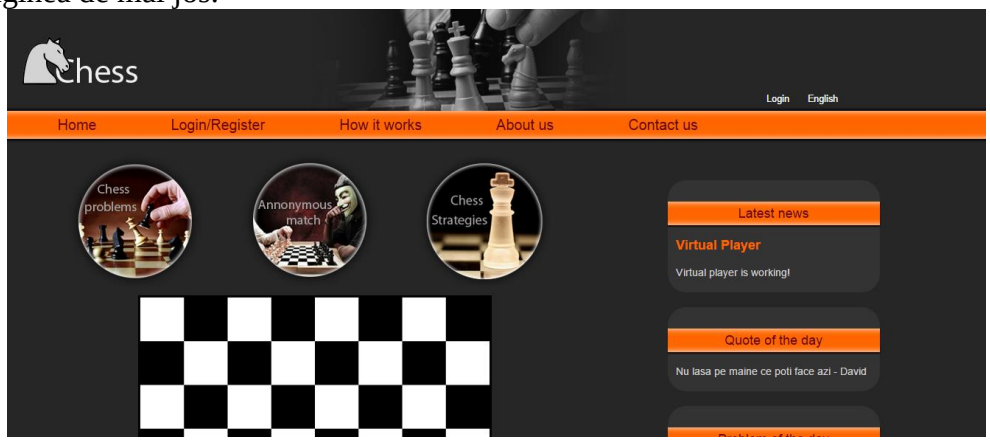

Figura 7.2 Interfața grafică a aplicației

7.2. Manual de utilizare

În continuare va fi prezentat manualul de utilizare al aplicației. În primul rând se va face o trecere în revistă a componentelor, urmând că mai apoi să se realizeze două acțiuni că și exemplu: jucărea unui meci de șah și autentificarea.

7.2.1. Componentele interfetei grafice

În imaginea de mai jos sunt prezentate principalele componente ale aplicației.

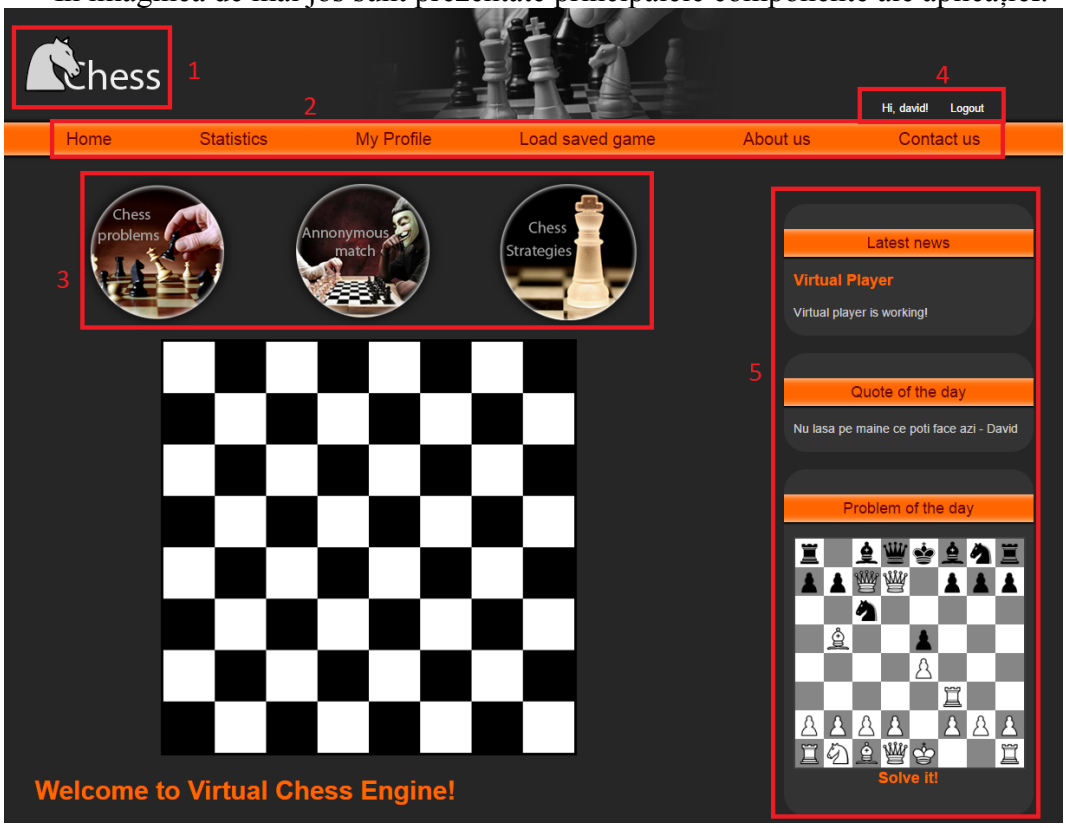

Figura 7.3 Principalele componente ale aplicației

1. Logoul aplicației
2. Meniu principal
3. Meniul secundar
4. Componenta de autentificare, responsabilă pentru autentificarea sau afișarea utilizatorului logat. Dacă utilizatorul nu este logat, se va afișa mesajul „login”, iar dacă utilizatorul este logat, se va afișa numele acestuia, împreună cu un link pentru logout.
5. Barele laterale, conțin problema de șah a zilei, citatul zilei și ultimele noutăți

7.2.2. Jucarea unui meci de șah

Pentru jucărea unui meci de șah, utilizatorul trebuie să navigheze la meniul secundar și să selecteze opțiunea pentru jucărea unui meci de șah, că în imaginea de mai jos:

Figura 7.4 Selectarea optiunii de jucare a unui meci de șah

Utilizatorul va selectă culoarea pieselor și nivelul de dificultate și va apăsa pe butonul de începere a meciului. Sistemul va redirecționa utilizatorul către tabla de șah:

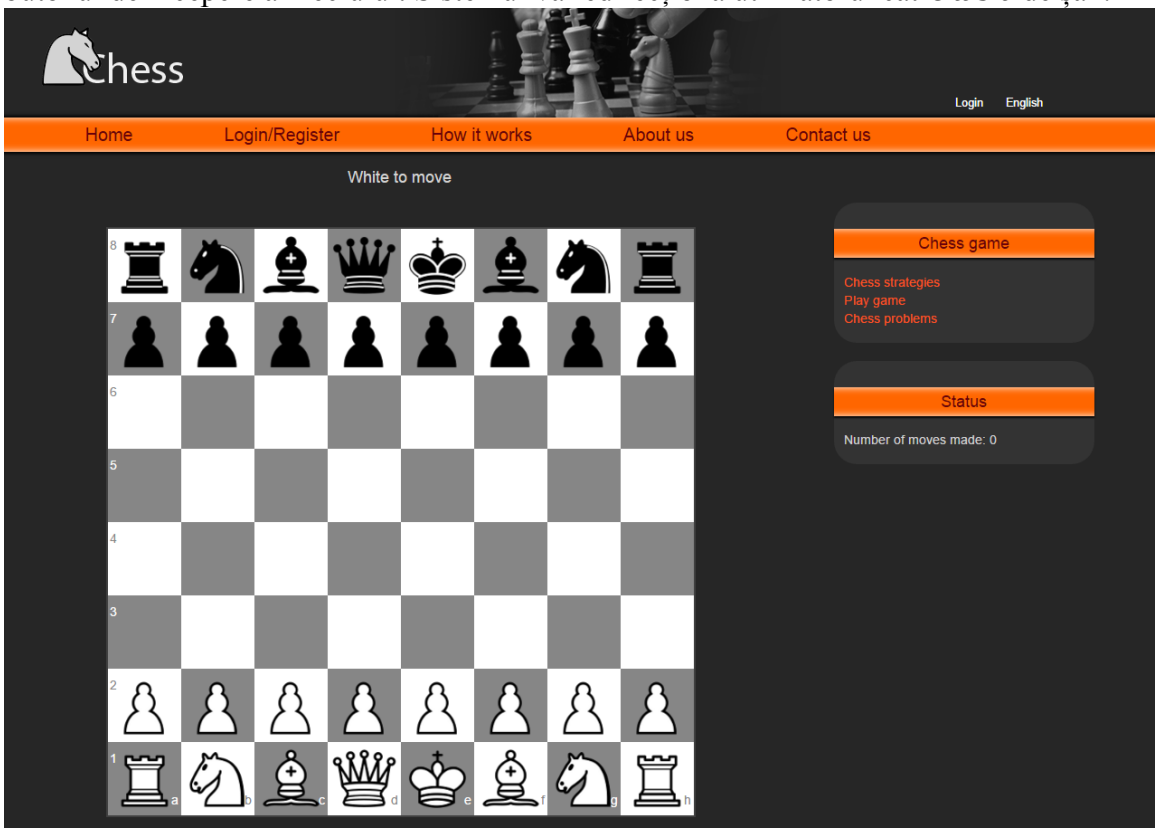

Figura 7.5 Selectarea opțiunii de jucare a unui meci de șah

De aici utilizatorul va face mutări pe tablă de șah și va aștepta răspunsul jucătorului virtual.

7.2.3. Autentificare

Pentru autentificare, utilizatorul trebuie să navigheze la componența de autentificare și să apese pe linkul de „login”, că în imaginea de mai jos:

Figura 7.6 Butonul de autentificare

Utilizatorul va fi redirecționat către o pagină ce conține un formular de autentificare, că în imaginea de mai jos:

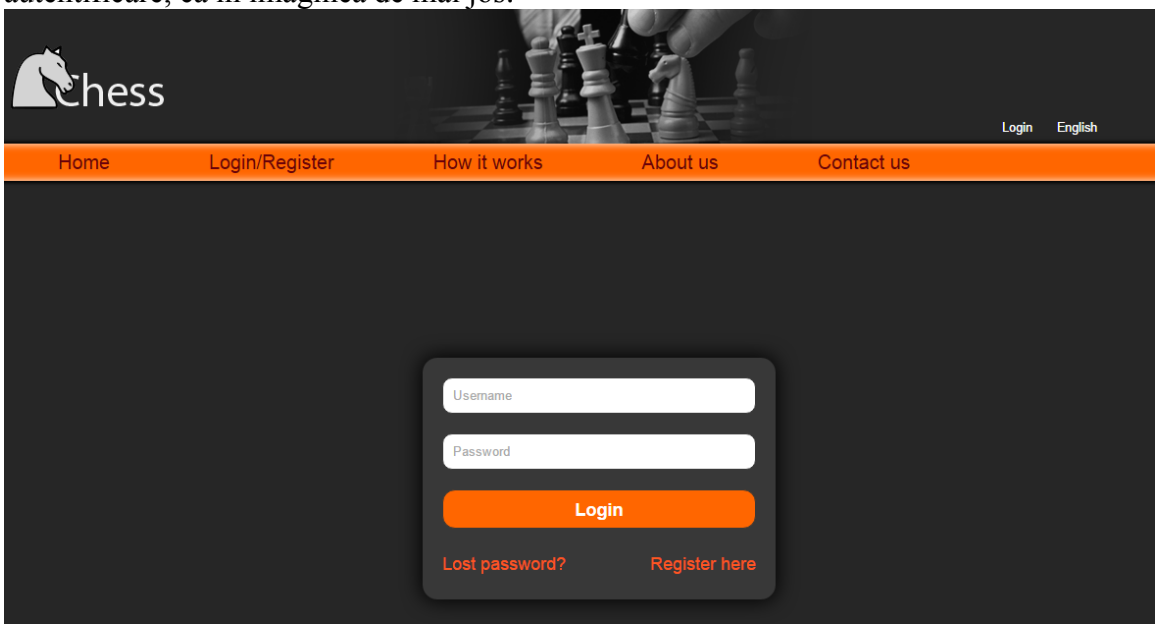

Figura 7.7 Formularul de autentificare

Pentru autentificare, utilizatorul introduce numele și parolă și apăsa pe butonul de login. Dacă autentificarea s-a realizat cu succes, în componența de autentificare va apărea numele utilizatorului, că în imaginea de mai jos:

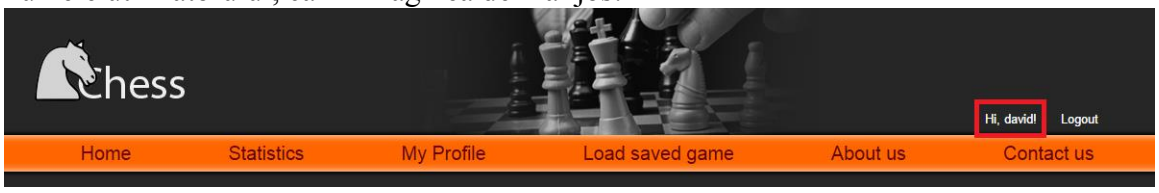

Figura 7.8 Componenta de autentificare

Capitolul 8. Concluzii

În acest capitol vor fi prezentate obiectivele ce au fost atinse prin acest proiect, precum și posibilitățile de dezvoltare ulterioare.

8.1. Realizarea obiectivelor propuse

Aplicația ChessEngine reușește să își atingă scopul, acela de a putea oferi atât suport jucătorilor profesioniști de șah, cât și mijloc de divertisment jucătorilor ocazionali. Obiectivele principale au fost atinse prin realizarea funcționalităților principale, acestea fiind descrise în continuare. Jucărea unui meci de șah se poate face alegând culoarea pieselor și un grad de dificultate, interacționând cu jucătorul virtual de șah. Vizualizarea strategiilor de șah se poate face accesând pagină de strategii, fiecare dintre acestea având o lista de mutări ce pot fi vizualizate pe tablă de șah, iar problemele de șah presupun parcurgerea pe tablă de șah a mutărilor ce duc la soluție. Aceste funcționalități principale sunt ușor accesibile, fiind disponibile pe toate paginile interfeței grafice.

8.2. Dezvoltări ulterioare

În continuare vor fi prezentate posibilitățile de dezvoltare ulterioare. Deoarece aplicația a fost dezvoltată utilizând o gamă largă de design pattern-uri, codul sursă este ușor de reutilizat, iar componentele sunt ușor de înlocuit.

Astfel, una dintre posibilitățile de dezvoltare ulterioară este externalizarea jucătorului virtual de șah într-un serviciu web separat, ce să poată rulă pe un cluster dedicat. Acest lucru poate fi făcut cu ușurință, iar arhitectură harware necesară este simplistă, prin faptul că jucătorul virtual nu are stări, astfel nefiind nevoie de o bază de date în spatele acestuia.

O altă posiblitate de dezvoltare ulterioară este includerea tehnicilor de machine learning în cadrul jucătorului virtual, în special pe partea funcției de evaluare. Astfel ar putea fi îmbunătățit jocul de deschidere al jucătorului virtual. O astfel de implementare ar putea folosi și conceptul de “dictionar”, o bază de date optimizată pentru citire, unde există perechi de tipul poziție pe tablă de șah și mutare recomandată pentru acea poziție.

Adăugarea posibilității de jucare a meciurilor de șah între utilizatori ar crește traficul în cadrul platformei, lucru ce poate fi realizat cu ușurință prin adoptarea unei arhitecturi peer to peer.

De asemenea, implementarea unui sistem prin care utilizatorii pot interacționa între ei prin intermediul mesajelor, al forumurilor de discuție și al liste de prieteni ar putea îmbunătăți experiență acestora.

Pentru interfață grafica, o imbunatarire ar fi dezvoltarea unor aplicații native pentru sistemele de operare mobile, cum ar fi Android, iOS sau Windows Phone. Acest lucru este posibil, comunicarea cu partea de backend a sistemului realizându-se cu ajutorul protocolului HTTP.

Bibliografie

[1] Craig Walls, Spring in Action (3rd Edition), Manning Publications, 2011

[2] Hamidreza Sattari, Shameer Kunjumohamed, Spring Web Services 2 Cookbook, Packt Publishing, 2012

[3] David Levy, Chess and Computers, Computer Science Press. 1976

[4] Eda Okur, Selin Kavuzlu, Development of an adaptive chess program, Boğaziçi University, 2011

[5] George Atkinson, Chess and Machine Intuition, Intellect, School of Art and Design, Earl Richards Road North, 1998

[6] David E. Welsh, Boris Baczynskyj, Computer Chess II, Wm. C. Brown, 1985

[7] Philippe Kruchten, The Rational Unified Process: An Introduction (3rd Edition), Addison-Wesley Professional, 2000

[8] Eric Freeman, Elisabeth Robson, Kathy Sierra, and Bert Bates, Head First Design Patterns, O'Reilly Media, Inc. 2004

[9] Ronald J. Leach, Introduction to Software Engineering, Second Edition (Chapman & Hall/CRC Innovations in Software Engineering and Software Development Series), CRC Press, 2014

[10] Chess programming wiki, disponibila la: https://chessprogramming.wikispaces.com/

[11] Documentatia oficiala Hibernate, disponibila la: http://hibernate.org/orm/documentation/5.1/

[12] Documentatia oficiala MySQL, disponibila la: https://dev.mysql.com/doc/

[13] Documentatia oficiala Spring, disponibila la: http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/

[14] Wikipedia, articol despre JavaScript, disponibil la: https://ro.wikipedia.org/wiki/JavaScript

[15] Documentatia oficiala ChessBoard.js, disponibila la: http://chessboardjs.com/

[16] Documentatia oficiala JSP, disponibila la: http://docs.oracle.com/javaee/5/tutorial/doc/bnajo.html

[17] Documentatia oficiala Java, disponibila la: https://docs.oracle.com/javase/tutorial/

[18] Documentatia oficiala Spring MVC, disponibila la: http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html

[19] Documentatia oficiala Spring Security, disponibila la: http://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/

[20] Documentatia oficiala GIT, disponibila la: https://git-scm.com/documentation

[21] Articol RESTful Web Services: The basics, disponibil la: http://www.ibm.com/developerworks/library/ws-restful/

[22] François Dominic Laramée, Articol Chess Programming Evaluation Functions, 2000, online, disponibil la:

http://www.gamedev.net/page/resources/_/technical/artificial-intelligence/chess-programming-part-vi-evaluation-functions-r1208

[23] Articol Chess Programming Basic search, disponibil la: http://www.gamedev.net/page/resources/_/technical/artificial-intelligence/chess-programming-part-iv-basic-search-r1171

Anexa 1 – Chestionarul de evaluare al aplicatiei

Cu ce scop folositi aceasta platforma?

- ○ Antrenament de sah
- ○ Divertisment

Cat de usor accesibile considerati ca sunt functionalitatile principale ale platformei (jucarea unui meci de sah, strategii si probleme de sah)?

- ○ Foarte greu
- ○ Greu
- ○ Mediu
- ○ Usor
- ○ Foarte usor

Cat de intuitiva este interfata grafica?

- ○ Foarte neintuitiva
- ○ neintuitiva
- ○ Medie
- ○ Intuitiva
- ○ Foarte intuitiva

Considerati ca design-ul interfetei grafice este discret?

- ○ Nu
- ○ Culorile si imaginile atrag atentia prea mult
- ○ Se poate si mai bine
- ○ Da

Considerati ca mesajele de eroare sunt suficient de usor de inteles?

- ○ Nu
- ○ Da
- ○ Nu stiu

Cat de usor recunoasteti starea in care se afla sistemul la un anumit moment?

- ○ Foarte greu
- ○ Greu
- ○ Mediu
- ○ Usor
- ○ Foarte usor

Cat de multumiti sunteti de timpul de raspuns al jucatorului virtual de sah cu urmatoarea mutare?

- ○ Foarte nemultumit
- ○ Nemultumit
- ○ Mediu
- ○ Multumit
- ○ Foarte multumit

Considerati utila ideea de a avea un cont pe aceasta platforma?

- ○ Da
- ○ Nu
- ○ Nu stiu

Pe o scara de la 1 la 5, ce nota ati da jucatorului virtual de sah din punct de vedere al calitatii mutarilor?

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5

Pe o scara de la 1 la 5, ce nota ati da interfetei grafice?

- ○ 1
- ○ 2
- ○ 3
- ◉ 4
- ○ 5

Ce imbunatatiri ati aduce platformei de sah?

Răspunsul dvs.

Anexa 2 – Glosar de termeni

API	Application Programming interface
Backend	Partea de logica a unui sistem, uneori denumit si server
CPU	Central Processing Unit
DB	Database
DI	Dependency injection
Frontend	Partea de interfata grafica a unui sistemm
HTTP	Hypertext Transfer Protocol
IoC	Inversion of Control
JSP	JavaServer Pages
JVM	Java Virtual Machine
MVC	Model - View - Controller
ORM	Object Relational Mapping
REST	Representational State Transfer
SGBD	Sistem de gestiune a bazelor de date
SQL	Structured Query Language
UML	Unified Modeling Language
URI	Unique resource identifier
URL	Uniform resource location
WS	Web Service