



UNIVERSITATEA DIN BUCUREŞTI



FACULTATEA
DE
MATEMATICĂ ȘI
INFORMATICĂ

SPECIALIZAREA INFORMATICĂ

Lucrare de licență

**CHESS WEB APPLICATION
UNICHESS**

**Absolvent
Bălăuță-Amargheoalei
Albert-Ionuț**

**Coordonator științific
Radu Eugen Boriga**

București, iunie 2024

Rezumat

Jocul de șah este renumit în toată lumea pentru natura sa competitivă și aspectele strategice pe care le cuprinde. În cadrul cercului nostru universitar, mulți studenți au dezvoltat o pasiune pentru acest joc datorită provocărilor mintale pe care le prezintă.

În acest sens, vom dezvolta o aplicație Web [1] similară cu Lichess [2] sau Chess.com [3], dar care să fie destinată exclusiv celor care studiază în cadrul Universității din București. Aplicația va conține jocul de șah implementat cu toate regulile sale ce le va permite studenților să joace între ei și va avea o interfață ușor de utilizat ce le va oferi posibilitatea de a crea jocuri, de a crea sau de a se alătura la turnee și de a deveni jucători mai buni prin analizarea meciurilor pentru a vedea ce mutări au greșit și ce mutări mai bune ar fi putut face.

Proiectul va fi realizat în limbajul Python [4] pentru dezvoltarea jocului și a backend-ului [6] în general, alături de framework-ul Django [5] pentru construirea aplicației Web. Pentru partea de frontend [6] se vor folosi HTML [7], CSS [8] și JavaScript [9].

Abstract

The game of chess is known worldwide for its strategic elements and its competitive nature. Within our university, many students have developed a passion for this game due to the intellectual challenges it presents.

To serve this purpose, we will develop a Web application [1] similar to Lichess [2] or Chess.com [3], but exclusively for those studying at the University of Bucharest. The application will feature the game of chess fully implemented along with all of its rules, allowing students to play against each other. It will also have a user-friendly interface that will enable them to create games, join and play in tournaments and improve their skills by analyzing their games to see what moves they made incorrectly and what better moves they could have made.

The project will be developed in Python [4] for the chess game and general backend [6] development, along with the Django [5] framework for building the web application. HTML [7], CSS [8] and JavaScript [9] will be used for the frontend [6].

Cuprins

1. Introducere.....	5
1.1 Motivație.....	5
1.2 Domenii abordate.....	5
1.3 Structura lucrării.....	6
1.3.1 Aplicația Web.....	6
1.3.2 Implementarea jocului de șah.....	6
1.3.3 Analiza unei partide.....	7
2. Aplicația Web.....	8
2.1. Introducere.....	8
2.1.1. Obiective principale.....	8
2.1.2. Experiența utilizatorului.....	9
2.2. Aspecte tehnice.....	10
2.2.1. Arhitectura.....	10
2.2.2. Limbaje de programare.....	10
2.2.3. Baza de date.....	11
2.3. Interacțiunea cu utilizatorul.....	12
2.3.1. Autentificarea.....	12
2.3.2. Profilul personal și clasamentul jucătorilor.....	15
2.4. Administrarea jocurilor și a turneelor.....	16
2.4.1. Funcționalități jocuri.....	16
2.4.2. Funcționalități turnee.....	18
3. Implementarea jocului de șah.....	23
3.1. Introducere reguli de joc.....	23
3.1.1. Informații generale.....	23
3.1.2. Mutatul pieselor.....	24
3.1.3. Mișcări speciale.....	25
3.2. Aspectul jocului.....	25
3.2.1. Construirea tablei de șah.....	25
3.2.2. Adăugarea pieselor cu imagini specifice.....	27
3.3. Funcționalități principale.....	30
3.3.1. Acțiunea de a muta o piesă.....	30

3.3.2. Mutarea corectă a pieselor.....	33
3.3.3. Situații speciale.....	36
3.3.4. Actualizare mutări în timp real.....	39
3.3.5. Administrarea timpului.....	42
3.4. Elemente finale.....	44
4. Analiza unei partide.....	47
4.1. Introducere.....	47
4.1.1. Descrierea funcționalității.....	47
4.2. Implementarea paginii.....	48
4.2.1. Structura paginii.....	48
4.2.2. Navigarea prin mutări.....	48
4.2.3. Evidențierea mutării curente.....	49
4.3. Integrare model AI.....	50
4.3.1. Interacțiunea cu modelul.....	50
4.3.2. Afisarea sugestiilor.....	51
5. Concluzie.....	53

Capitolul 1

Introducere

1.1 Motivație

Jocul de șah a devenit o pasiune pentru mine încă din liceu, iar posibilitatea de a putea juca un meci online, la orice oră, doar prin apăsarea câtorva click-uri, a fost un ajutor considerabil. Posibilitatea de a analiza jocuri m-a ajutat să progresez și, prin urmare, am obținut și o calificare la Olimpiada Națională de Șah.

Construirea unui joc atât de complex reprezintă o provocare îndeajuns de dificilă pe care mi-am dorit să o fac dintotdeauna. Această experiență reprezintă o oportunitate excelentă pentru a înfățișa toate cunoștințele învățate pe parcursul acestor trei ani. Cu atât mai mult, șansa de a crea și de a crește o comunitate locală de pasionați de șah în cadrul studentilor contribuie semnificativ la motivația mea de a face un proiect de succes.

1.2 Domenii abordate

- . Lucrarea de licență cuprinde următoarele domenii principale:
 - **Dezvoltare Web.** Proiectul oferă utilizatorilor o interfață ușor de utilizat pentru a naviga în diferite secțiuni ale aplicației: profil jucător, clasament jucători, turnee, analiză meci.
 - **Algoritmică.** Aplicația necesită dezvoltarea unor algoritmi eficienți pentru a valida mișcări generale, precum și situații speciale cum ar fi: rocada, promovarea unui pion și detectarea că regele este în „șah” [\[10\]](#).
 - **Game Design.** Jocul cuprinde o interfață atractivă ce le va facilita înțelegerea pieselor, cum ar putea face mutări și alte acțiuni.

1.3 Structura lucrării

Lucrarea este structurată în 3 capitole principale:

- **Aplicația Web.** Include procesul de creare a aplicației, ce cuprinde informații despre arhitectură, diagrama bazei de date, îmbunătățirea aspectului în frontend [6] și implementarea funcționalităților în backend [6].
- **Implementarea jocului de șah.** Cuprinde toate detaliile referitoare la introducerea regulilor de joc și validarea tuturor mutărilor.
- **Analiza unei partide.** Descrie secțiunea dedicată analizei unui meci prin recomandarea de mutări și punerea în evidență a mutărilor greșite ce au avut posibilitatea să dezechilibreze soarta meciului.

1.3.1 Aplicația Web

Pentru a realiza o aplicație web [1] complet funcțională vom începe prin a determina funcționalitățile generale necesare pentru a susține o platformă online ce permite utilizatorilor să joace șah. În acest sens, vom avea o parte dedicată interacțiunii cu utilizatorul prin implementarea posibilității de autentificare în aplicație și vizualizare a unui profil personal ce va conține diferite statistici relevante experienței sale. În plus, vom furniza posibilitatea de a juca cu alte persoane prin intermediul unui sistem de administrare a jocurilor și a turneelor.

1.3.2 Implementarea jocului de șah

Pentru a realiza un joc de șah online complet funcțional care să cuprindă toate regulile sale, vom începe prin a determina aspectul tablei de șah și a pieselor de joc. Astfel, vom adăuga funcționalitățile esențiale de a face click și „drag and drop” pentru a putea muta piesele. După acestea, vom implementa modalitatea de mișcare a fiecărei piese, făcând toate validările necesare pentru a ne asigura că jocul funcționează corespunzător, împreună cu mutările speciale ale jocului.

La sfârșit, vom introduce niște elemente finale ce țin de parcursul generală a unui meci și modurile sale de încheiere: alb și negru să mute o singură dată pe rând, derularea corectă a timpului, opțiunea de a renunța sau de a oferi o remiză.

1.3.3 Analiza unei partide

Pentru a completa proiectul nostru, vom adăuga o funcționalitate ce îi va permite utilizatorului să facă o analiză a partidelor de șah. Analiza partidelor joacă un rol important în dezvoltarea jucătorilor, motiv pentru care este esențial să oferim și această unealtă. În acest sens, vom adăuga un buton în cadrul meciurilor finalizate ce ne va redirecționa către o pagină dedicată analizei jocului.

În cadrul acestei secțiuni, utilizatorul va avea la dispoziție o listă a mutărilor parcuse în meci, tabla de șah ce va înfățișa corect starea partidei și o listă de sugestii împreună cu o notă de evaluare a poziției curente. În plus, cele mai bune 3 sugestii vor fi ilstrate și pe tablă prin atașarea unor săgeți corespunzătoare. Pentru a genera sugestiile și scorul poziției, vom integra un model AI [11] puternic ce va fi capabil să gestioneze complexitatea acestui joc cu o capacitate mult mai dezvoltată decât cea a unui om.

Capitolul 2

Aplicația Web

2.1. Introducere

Pentru a dezvolta aplicația web [1], trebuie să definim precis obiectivele noastre principale și să determinăm cum ar arăta parcursul utilizatorului nostru pe platforma noastră într-o manieră eficientă și intuitivă.

2.1.1. Obiective principale

Obiectivele noastre trebuie stabilite corespunzător pentru a construi o aplicație funcțională și atractivă pentru a convinge un student că acesta este locul cel mai potrivit pentru a juca șah cu alți colegi și pentru a-și dezvolta împreună abilitățile de a practica acest sport.

În acest sens, următoarele sunt obiectivele principale în vederea dezvoltării platformei noastre:

- Crearea unei aplicații web [1] intuitive și ușor de utilizat pentru jucătorii de șah de toate nivelurile.
- Implementarea unui sistem de autentificare securizat pentru utilizatori.
- Oferirea posibilității de a crea și administra partide de șah și turnee.
- Furnizarea unui profil personal ce conține statistici relevante despre utilizator și istoric al meciurilor sale.
- Integrarea unui sistem de clasare a jucătorilor prin implementarea unui punctaj universal ce reflectă nivelul de joc (ELO [12]).
- Navigare fluidă prin pagini și fără erori.

2.1.2. Experiența utilizatorului

Utilizatorii vor avea acces la o varietate de pagini structurate astfel încât parcurgerea aplicației noastre să fie cursivă și intuitivă. În acest sens, platforma noastră va fi împărțită în felul următor:

- **Autentificare:** secțiunea aceasta va fi reprezentată de o pagină de înregistrare unde utilizatorul va putea să-și creeze un cont nou și o pagină de autentificare ce îi va permite să acceseze diferitele funcționalități ale aplicației.
- **Pagina principală:** acesta este locul unde utilizatorii vor fi redirecționați imediat după autentificare și cuprinde o listă cu 5 opțiuni: creare joc nou, jocuri active, turnee, profilul personal și clasament.
- **Creare și vizualizare jocuri:** partea de creare constă în prima opțiune din pagina principală ce trimită utilizatorul către o pagină de creare a unui joc nou, unde acesta va putea să-și aleagă oponentul și opțiunile de timp ale jocului, după care va fi redirecționat către pagina de joc. Partea de vizualizare constă în a doua opțiune din pagina principală ce va înfățișa o listă a tuturor jocurilor active.
- **Interacțiunea cu turneele:** secțiunea aceasta poate fi vizualizată prin accesarea opțiunii a treia din pagina principală. Cuprinde o listă a turneelor active și un buton de creare a unui turneu nou. Secțiunea de creare oferă utilizatorului posibilitatea de a face o nouă competiție cu diferite opțiuni la alegere. Vizualizarea este diferită în funcție de starea turneului. Dacă turneul este neînceput, jucătorii pot vedea participanții înregistrați la turneu și au opțiunea de a se alătura sau de a părăsi turneul înainte de începere. În plus, dacă utilizatorul este și creatorul competiției, acesta va putea adăuga sau șterge participanți la alegere. În cazul în care turneul a început, pagina va înfățișa clasamentul jucătorilor înregistrați și o listă a partidelor pe runde.
- **Clasarea jucătorilor:** este a patra opțiune din pagina principală și prezintă un clasament al tuturor jucătorilor înregistrați în funcție de ELO [\[12\]](#).
- **Profilul personal:** reprezintă ultima opțiune din pagina principală. Conține statistici relevante despre jucător cum ar fi: punctajul universal (ELO [\[12\]](#)), numărul de jocuri, numărul de turnee jucate și câștigate, victorii, egaluri, înfrângeri, jucând cu piesele albe sau cu cele negre.

2.2. Aspecte tehnice

În acest capitol, vom prezenta detalii tehnice despre cum vom construi platforma noastră. Astfel, vom oferi amănunte despre arhitectura aplicației, limbajele de programare folosite și structura bazei de date.

2.2.1. Arhitectura

Django [5] este un framework gratis și open-source construit cu scopul de a permite pasionaților de programare să dezvolte aplicații web. Arhitectura aplicației noastre este bazată pe acest framework, urmând modelul MTV (Model-Template-View [13]) cu scopul de a avea o separare clară a componentelor aplicației.

- **Model (M)**: se referă la structura informațiilor și definirea entităților bazei de date. Se ocupă de toate operațiunile legate de prelucrarea datelor, precum interogări și actualizări corespunzătoare, în funcție de acțiunile făcute de utilizator.
- **Template (T)**: reprezintă partea de prezentare a aplicației (frontend [6]). Un template reprezintă un fișier HTML [7], însă cu ajutorul acestui framework, putem customiza mai bine aspectul paginilor prin folosirea unor tag-uri specifice.[14]

```
<h1>My Homepage</h1>
<p>My name is {{ firstname }}.</p>
```

- **View (V)**: conține dezvoltarea logicii din spate a aplicației (backend [6]). Similar unui controller [15], gestionează cererile făcute de utilizatori, interacționează cu baza de date și întoarce răspunsuri corespunzătoare oferind un context cu mai multe informații ce pot fi accesate în cadrul template-urilor.

2.2.2. Limbaje de programare

Aplicația noastră folosește o serie de limbaje de programare pentru a asigura funcționarea corectă și aspectul plăcut al paginilor.

Pe partea de backend, deoarece folosim framework-ul Django[5], vom dezvolta logica de server ce cuprinde funcționalitățile principale ale aplicației cu ajutorul limbajului Python [4]. Aceasta este un limbaj de programare interpretat și de înalt nivel ce pune accent pe lizibilitatea și simplitatea codului, având o sintaxă clară cu multe structuri de date implementate ce permite programatorilor să-și construiască aplicații mult mai ușor de înțeles

și de întreținut.

Pe partea de frontend, vom folosi 3 limbaje cu trăsături diferite în vederea înfrumusețării aspectului vizual al aplicației. HTML [\[7\]](#) este un limbaj de marcare ce va fi folosit pentru structurarea unei pagini web. CSS [\[8\]](#) va fi utilizat cu scopul de a stiliza diferite secțiuni ale paginilor și pentru a face aplicația cât mai atractivă. JavaScript [\[9\]](#) va fi folosit cu scopul de a dinamiza interacțiunea utilizatorului.

2.2.3. Baza de date

Pentru crearea bazei de date, aplicația va utiliza SQLite [\[16\]](#), un sistem de gestionare a bazelor de date ușor de folosit datorită simplității sale și datorită faptului că nu necesită un server de baze de date separat pentru a ține informații.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

În acest sens, entitățile aplicației noastre sunt următoarele:

- **Users**: entitate predefinită în Django [\[5\]](#) ce stochează informațiile de autentificare (username, parolă) și câteva date personale (nume, prenume, email).
- **Profile**: o extensie a entității Users, conține o cheie externă către această tabelă și prezintă statistici despre partidele jucate de utilizator, precum și ELO-ul [\[12\]](#) sau, informații ce vor fi afișate pe pagina de profil personal.
- **Game**: entitate ce stochează informații despre fiecare joc de șah. Conține o cheie externă dublă către tabela Users, semnificând jucătorul cu piesele albe și cel cu piesele negre. În cazul în care partida se desfășoară în cadrul unui turneu, atunci vom mai avea câte o cheie externă către tabelele Tournaments și Rounds.
- **Tournaments**: această tabelă conține date specifice despre formatul turneelor și are o cheie externă către tabela de Users pentru a săi cine este fondatorul turneului.
- **Rounds**: entitate ce păstrează informații despre fiecare rundă a unui turneu.

2.3. Interacțiunea cu utilizatorul

Interacțiunea cu utilizatorul în aplicația noastră este esențială pentru a asigura o experiență plăcută și intuitivă. În cadrul acestui capitol, vom detalia modul în care utilizatorul interacționează cu platforma noastră, prin procesul de autentificare și furnizarea unui profil personal cu diverse statistici despre activitatea sa.

Pentru început, vom adăuga o imagine de fundal generată cu AI [11] pentru toate paginile cu scopul de a oferi un aspect vizual atractiv și pentru a captiva utilizatorii.



Figura 2.1 - Imaginea de fundal

2.3.1. Autentificarea

Autentificarea utilizatorilor este împărțită în două secțiuni principale: înregistrare și logare.

Pagina de înregistrare permite utilizatorilor să-și creeze propriul cont la care doar ei vor avea acces prin completarea unor informații de bază precum username-ul, numele de familie, prenumele, adresa de email și parola. Această funcționalitate se va realiza prin intermediul crearea unui formular ce va extinde clasa UserCreationForm [17] din Django [5]. Formularul se va numi „SignUpForm”, va defini câmpurile necesare pentru înregistrare și va conține o validare pentru adresa de email.

```

class SignUpForm(UserCreationForm):
    email = forms.EmailField(max_length=254, help_text='Required. Inform a valid email address.')
    first_name = forms.CharField(max_length=30, required=True, help_text = 'Required.')
    last_name = forms.CharField(max_length=30, required=True, help_text = 'Required.')

class Meta:
    model = User
    fields = ('username', 'first_name', 'last_name', 'email', 'password1', 'password2')

def clean_email(self):
    email = self.cleaned_data.get('email')
    if User.objects.filter(email=email).exists():
        raise forms.ValidationError("Email already exists")
    return email

```

Procesarea înregistrării se va realiza prin metoda „register” din fișierul „views.py”. În cazul în care primim o cerere de tip „POST” [18] și avem date valide, atunci vom autentifica utilizatorul și îl vom redirecționa către pagina principală. În caz contrar, îl vom păstra pe pagina de înregistrare.

```

def register(request):
    if request.method == 'POST':
        form = SignUpForm(request.POST)
        if form.is_valid():
            form.save()
            username = form.cleaned_data.get('username')
            raw_password = form.cleaned_data.get('password1')
            user = authenticate(username=username, password=raw_password)
            login(request, user)
            return redirect('home')
    else:
        form = SignUpForm()
    return render(request, 'registration/register.html', {'form': form})

```

După procesarea datelor prin template-ul „register.html”, aşa va arăta pagina noastră de înregistrare.

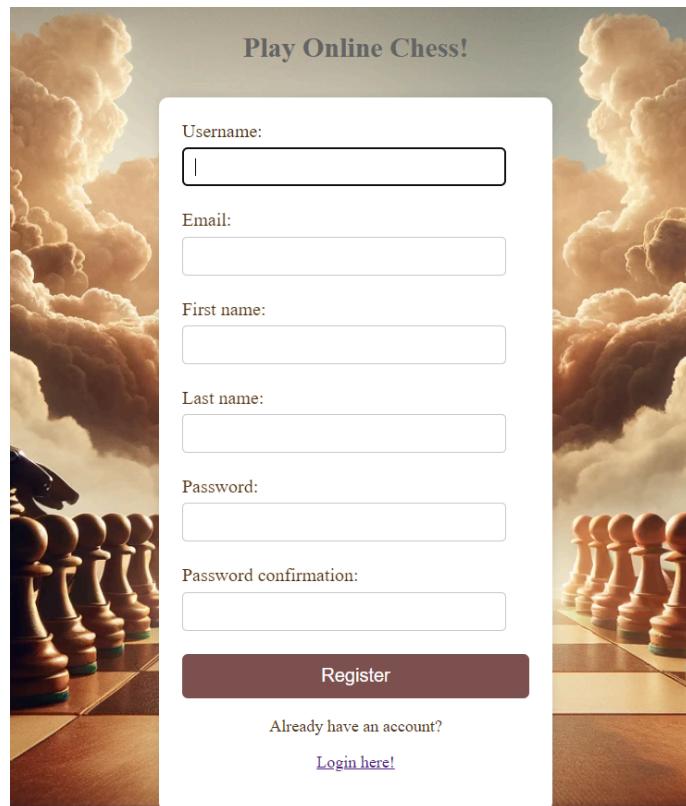


Figura 2.2 - Pagina de înregistrare

În ceea ce privește logarea utilizatorului pe platforma noastră, vom folosi logica deja existentă furnizată de Django [5] în pachetul „django.contrib.auth” [19].

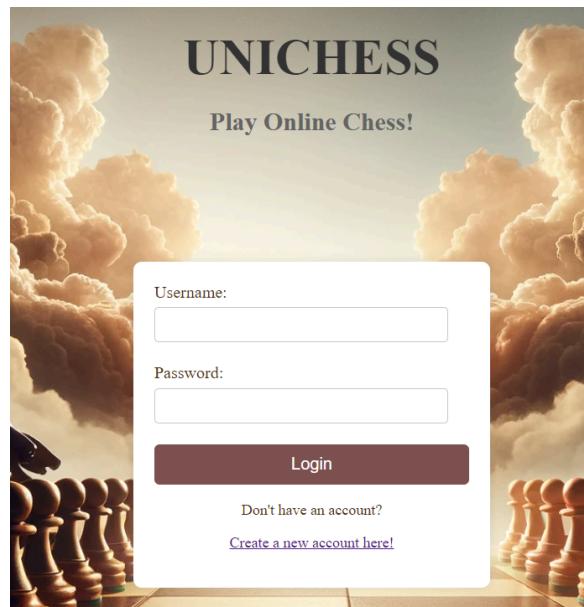


Figura 2.3 - Pagina de autentificare

2.3.2. Profilul personal și clasamentul jucătorilor

Profilul personal al utilizatorului reprezintă o secțiune în care se pot vizualiza diferite statistici despre activitatea sa pe platforma noastră. Implementarea acestei pagini constă în crearea entității Profile, împreună cu interogarea și actualizarea ei în funcție de rezultatele partidelor, procesarea corectă a informațiilor și afișarea lor frumos în template-ul „profile.html”. În plus, această pagină va conține și o listă a ultimelor partide jucate de utilizator.

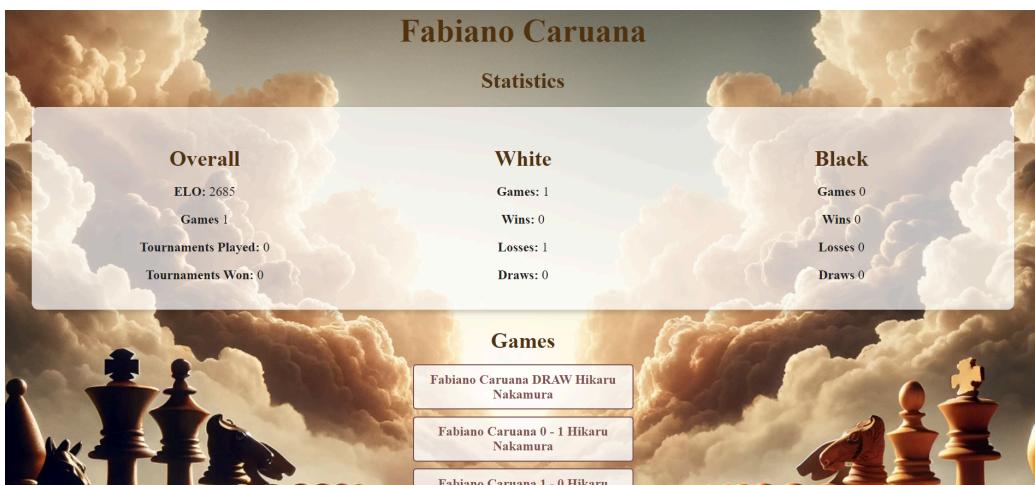


Figura 2.4 - Pagina de profil

Pentru a implementa pagina cu clasamentul, vom selecta într-o listă toți jucătorii înregistrați pe platformă și vom face o sortare după punctajul lor universal (ELO [12]). După asta, vom transmite această informație drept context pentru template-ul „leaderboard.html”.

A screenshot of a leaderboard page. The background shows a chessboard with pieces. The title "Leaderboard" is at the top. A table lists the top 8 players with their ELO scores: Garry Kasparov (2715), Hikaru Nakamura (2700), Ian Nepomniatchi (2700), Magnus Carlsen (2700), Fabiano Caruana (2685), Anish Giri (2600), Balázs Albert (1500), and Vlad Andrei (1200).

Figura 2.5 - Clasamentul jucătorilor

2.4. Administrarea jocurilor și a turneelor

În această secțiune, vom oferi o perspectivă detaliată asupra modului în care utilizatorii pot interacționa cu partidele și turneele din cadrul aplicației. Vom explora funcționalitățile principale legate de crearea și vizualizarea unei partide de șah, precum și gestionarea turneelor începute sau neînceput, din perspectiva de organizator sau din perspectiva de participant.

2.4.1. Funcționalități jocuri

Vizualizarea jocurilor active presupune posibilitatea utilizatorilor de a observa lista meciurilor în desfășurare, cu posibilitatea de a participa drept spectator la un meci de șah. Aceasta poate fi accesată din pagina de început prin apăsarea butonului „Games” ce va redirecționa către pagina cu lista partidelor. Implementarea este una standard și constă în procesarea tuturor jocurilor active într-o listă și transmiterea acesteia drept context către template-ul „games_list.html”, care are ca scop afișarea partidelor. Dacă utilizatorul apasă click pe orice meci, atunci acesta va fi trimis să vizualeze ca spectator jocul în desfășurare.

```
{% extends "base.html" %}

{% block title %} Games {% endblock %}

{% block content %}


# Active Games




{% for game in games %}
  {%if game.isActive %}
    <li>
      <a href="{% url 'game_play' game.id %}">
        {{ game }}
      </a>
    </li>
  {%endif%}
  {%endfor%}
</ul>


{% endblock %}
```



Figura 2.6 - Lista cu partide în desfășurare

Procesul de creare a unui joc este un aspect esențial al aplicației noastre web și implică mai mulți pași. Utilizatorul trebuie să selecteze care va fi oponentul său și care vor fi parametrii de timp ai partidei. Implementarea este simplă și constă în crearea unui formular numit „GameForm” ce va implementa clasa „ModelForm” din pachetul „django.forms” [\[20\]](#) și procesarea corespunzătoare a acestuia de către server în cadrul metodei „create” din fișierul „views.py”.

```
class GameForm(ModelForm):
    class Meta:
        model = Game
        fields = ['white', 'black', 'duration', 'increment']
        widgets = {
            'duration': TextInput(attrs={"type": "number", "min": "1"}),
            'increment': TextInput(attrs={"type": "number", "min": "0"}) }

@login_required
def create(request):
    if request.method == "POST":
        form = GameForm(request.POST)
        if form.is_valid():
            game = form.save(commit=False)
            game.white_time_remaining = game.duration * 60
            game.black_time_remaining = game.duration * 60
            game.isActive = 1
            game.save()
            return redirect("game_play", game_id=game.id)
    else:
        form = GameForm()
    return render(request, 'games/create.html', {"form": form})
```

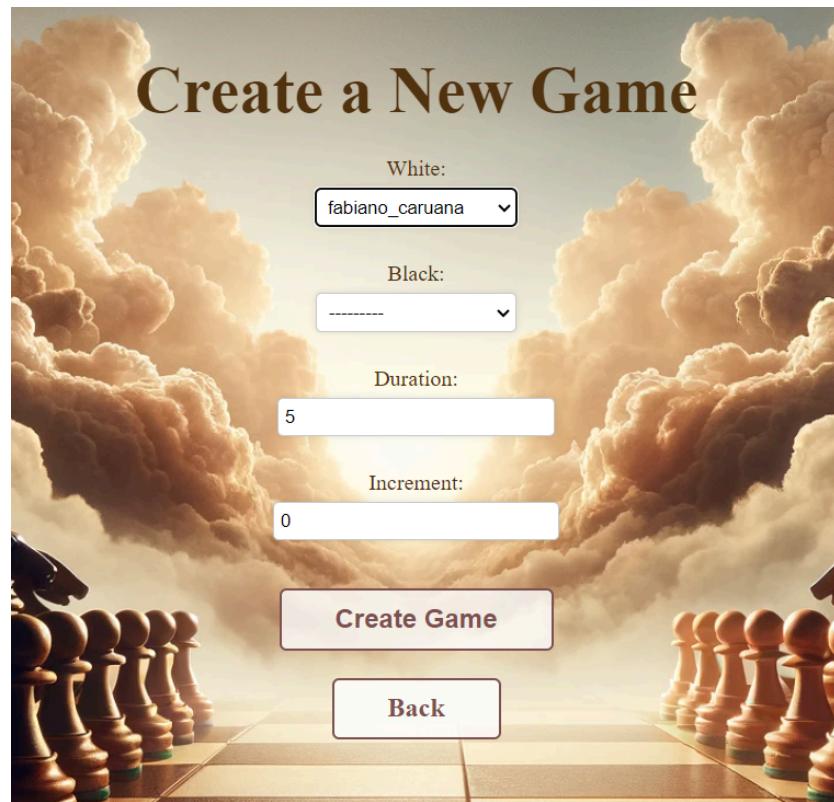


Figura 2.7 - Crearea unui joc nou

2.4.2. Funcționalități turnee

Pentru a vizualiza lista turneelor, vom proceda în mod similar cu vizualizarea listei de jocuri. Dupa ce va apăsa butonul „Tournaments” din pagina principală, utilizatorul va avea o listă de turnee active pe care le va putea accesa și un buton de creare a unui nou turneu.

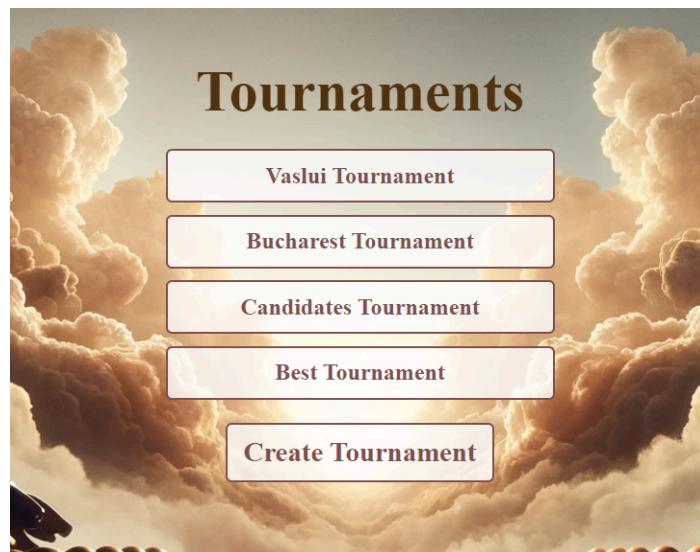


Figura 2.8 - Lista turneelor

Procesul de creare a unui turneu este la fel ca cel de creare a unei partide de şah. Singura diferenţă constă în furnizarea unor date diferite cum ar fi: numele turneului, numărul minim şi maxim de jucători, parametrii de timp şi premiul.

În cadrul unui turneu neînceput, vizualizarea paginii va cuprinde mai multe informaţii utile şi o listă cu toţi participanţii înscrişi la momentul actual. În plus, se va pune la dispoziţie şi o serie de butoane ce va permite utilizatorilor să se alăture si organizatorului să adauge şi să eliminate jucători, precum şi posibilitatea să înceapă turneul mai devreme decât data de începere.

```
<h1>{{ tournament.name }}</h1>
<div class ="tournament-info">
  <p><strong>Creator:</strong> {{tournament.owner.first_name}} {{tournament.owner.last_name }}</p>
  <p><strong>Start Date:</strong> {{ tournament.start_date }}</p>
  <p><strong>Prize:</strong> {{ tournament.prize }} $</p>
  <p><strong>Minimum Players:</strong> {{ tournament.minimum_players }}</p>
  <p><strong>Maximum Players:</strong> {{ tournament.maximum_players }}</p>
  <p><strong>Players Joined:</strong></p>
  <ul>
    {% for player in tournament.players %}
      <li>{{ player }}</li>
    {% empty %}
      <li>No players have joined yet.</li>
    {% endfor %}
  </ul>
</div>
```

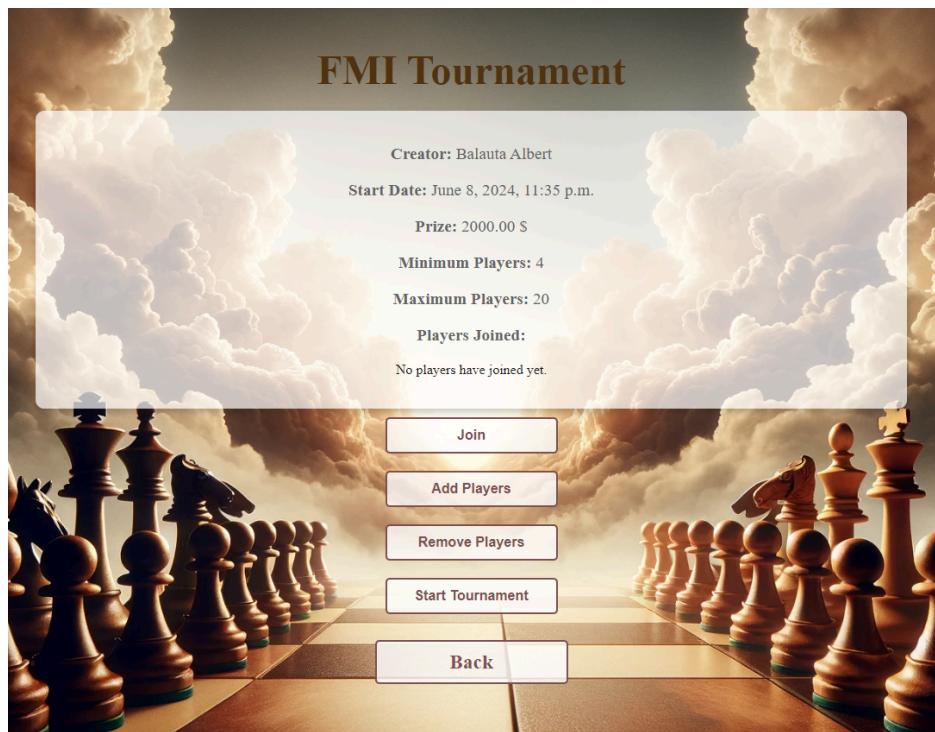


Figura 2.9 - Pagina unui turneu neînceput

Alăturarea la un turneu este o acțiune ce poate fi făcută de orice utilizator ce vizitează pagina de vizualizare. Acest lucru se întâmplă atunci când se apasă butonul de „Join”. Implementarea este una simplă și presupune procesarea în backend [6] prin metoda „join_tournament” din fișierul „views.py”.

```
def join_tournament(request, tournament_id):
    tournament = get_object_or_404(Tournament, id=tournament_id)
    if request.method == 'POST':
        if len(tournament.players) < tournament.maximum_players:
            tournament.players.append(request.user.username)
            tournament.save()
            return redirect('tournament_info', tournament.id)
        else:
            return render(request, 'tournaments/info_tournaments.html', {'tournament': tournament, 'error': 'Maximum number of players reached'})
    return redirect('tournament_info', tournament.id)
```

În cazul în care utilizatorul are rolul de creator al unui turneu, acesta are la dispoziție niște acțiuni adiționale pentru administrarea turneului său. Funcționalitatea de adăugare a participanților se realizează printr-un formular care este procesat de către server pentru a actualiza lista de participanți la turneu prin metoda „add_players” din fișierul „views.py”

```
@login_required
def add_players(request, tournament_id):
    tournament = get_object_or_404(Tournament, id=tournament_id)
    if request.user != tournament.owner:
        return redirect('tournaments')
    if request.method == 'POST':
        form = AddPlayerForm(request.POST, tournament=tournament)
        if form.is_valid():
            user = form.cleaned_data['user']
            if user.username not in tournament.players:
                tournament.players.append(user.username)
                tournament.save()
                return redirect('tournament_info', tournament.id)
        else:
            form = AddPlayerForm(tournament=tournament)
    return render(request, 'tournaments/add_players.html', {'form': form, 'tournament': tournament})
```

Analog, funcționalitatea de eliminare participanți este exclusiv disponibilă organizatorului turneului, iar implementarea sa prin metoda „remove_players” este foarte similară cu cea a adăugării de utilizatori.

```

@login_required
def remove_player(request, tournament_id):
    tournament = get_object_or_404(Tournament, id=tournament_id)
    if request.user != tournament.owner:
        return redirect('tournament_info', tournament_id=tournament_id)
    if request.method == 'POST':
        form = RemovePlayerForm(request.POST, tournament=tournament)
        if form.is_valid():
            user = form.cleaned_data['user']
            tournament.players.remove(user.username)
            tournament.save()
            return redirect('tournament_info', tournament_id)
    else:
        form = RemovePlayerForm(tournament=tournament)
    return render(request, 'tournaments/remove_players.html', {'form': form, 'tournament': tournament})

```

În continuare, organizatorul mai are posibilitatea și de a începe turneul când dorește el prin apăsarea butonului „Start Tournament”. Implementarea acestei funcționalități constă în schimbarea stării turneului prin `tournament.is_active = True` și generarea corectă a rundelor și a partidelor ce urmează să se desfășoare prin apelarea metodei „RoundRobinGeneration”. După prelucrarea acestor date, se va trimite contextul corespunzător pentru ca datele să fie afișate cu un aspect vizual plăcut în cadrul template-ului „active_tournament.html”.

```

@login_required
def start_tournament(request, tournament_id):
    tournament = get_object_or_404(Tournament, id=tournament_id)
    if request.user == tournament.owner and not tournament.is_active:
        players = [User.objects.get(username=player) for player in tournament.players]
        generated_rounds = RoundRobinGeneration(players)
        round_counter = 1
        for matches in generated_rounds:
            round_obj = Round.objects.create(tournament=tournament, round_number=round_counter)
            for white, black in matches:
                Game.objects.create(
                    white=white,
                    black=black,
                    duration=tournament.duration,
                    increment=tournament.increment,
                    round=round_obj,
                    tournament=tournament
                )
            round_counter += 1
        tournament.is_active = True
        tournament.save()
    return redirect('tournament_info', tournament.id)

```

```

def RoundRobinGeneration(players):
    n = len(players)
    rounds = []
    if n % 2 == 1:
        n += 1
        players.append(None)
    for round_num in range(n - 1):
        round_matches = []
        for i in range(n // 2):
            player1 = players[i]
            player2 = players[n - i - 1]
            if player1 is not None and player2 is not None:
                round_matches.append((player1, player2))
        players.insert(1, players.pop())
        rounds.append(round_matches)
    return rounds

```

După începerea turneului, pagina de vizualizare se va schimba și va fi împărțită în două secțiuni: clasamentul turneului și lista rundelor.

The screenshot shows a web page for the 'Candidates Tournament'. At the top, it says 'Candidates Tournament'. Below that is a 'Rankings' section with a table:

Rank	Player	Points
1	Hikaru Nakamura	1.5
2	Garry Kasparov	1.5
3	Fabiano Caruana	1
4	Magnus Carlsen	1
5	Anish Giri	1
6	Ian Nepomniatchi	0

Below the rankings is a 'Rounds' section titled 'Round 1 of Candidates Tournament' with three results:

- Fabiano Caruana 0 - 1 Anish Giri
- Hikaru Nakamura DRAW Garry Kasparov
- Ian Nepomniatchi 0 - 1 Magnus Carlsen

Figura 2.10 - Pagina unui turneu activ

Capitolul 3

Implementarea jocului de șah

3.1. Introducere reguli de joc

Șahul [10] este unul din cele mai vechi și populare jocuri de societate, practicat de 2 oponenți pe o tablă în carouri cu piese ce au un design special, de culori contrastante, de obicei alb și negru. Jucătorul cu piesele albe mută primul, după care se alternează mutările conform unor reguli fixe, fiecare încerând să captureze piesa principală a adversarului, Regele.

3.1.1. Informații generale

Jocul se desfășoară pe o tablă împărțită în 64 de pătrate dispuse pe 8 rânduri și 8 coloane. Indiferent de culorile actuale ale tablei de joc, pătratele deschise la culoare mai sunt denumite drept „alb”, iar cele închise drept „negru”. La început, se pun pe tablă 16 piese albe și 16 piese negre astfel încât fiecare jucător are la dispoziție setul său de piese.(Figura 3.1) [21]

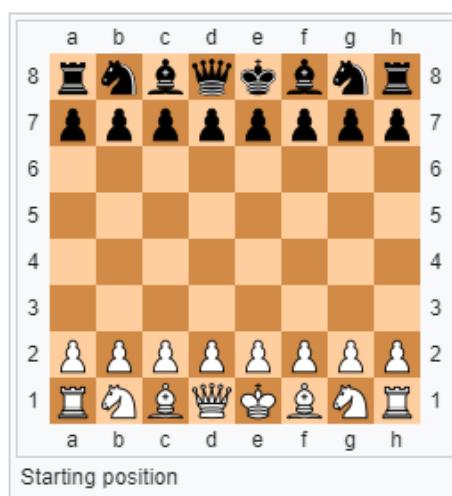


Figura 3.1 - Tablă de șah standard

3.1.2. Mutatul pieselor

Fiecare jucător controlează 16 piese ce sunt împărțite în felul următor: (Figura 3.2) [21]

Piece	King	Queen	Rook	Bishop	Knight	Pawn
Number of pieces	1	1	2	2	2	8
Symbols						

Figura 3.2 - Pieele de șah

Fiecare tip de piesă are metoda ei proprie de mișcare. O piesă se poate muta pe un pătrat liber sau în locul unei piese adverse, acțiune cunoscută drept capturarea unei piese. Când o piesă este capturată, aceasta este eliminată complet din joc. Cu excepția mutării calului și a rocadei, o piesă nu poate sări peste alte piese.

Urmează să explicăm cum se mișcă fiecare piesă, în ordinea în care sunt prezentate în Figura 2:

- Regele se poate muta exact un pătrat pe orizontală, verticală și pe diagonală. Acesta nu se poate afla la distanță de un pătrat față de regele adversar și nu se poate muta pe un pătrat controlat de oponent (adică un pătrat unde oponentul ar putea muta tura următoare). Nu poate fi capturat, iar atunci când este atacat se mai spune că este în „șah” (adică oponentul poate muta tura viitoare în locul în care se află regele) și, dacă acesta nu are posibilitatea să plece din acel loc, atunci este în „șah mat”, iar meciul se încheie. Împreună cu tura, aceste pot face o mutare specială, numită și rocadă, ce va fi explicitată în subsecțiunea 3.1.3
- Regina este considerată cea mai puternică piesă. Aceasta se poate muta în linie dreaptă, pe orice pătrat dorește jucătorul, pe toate direcțiile: orizontală, verticală și pe diagonală.
- Tura se poate muta în linie dreaptă pe orice pătrat, doar pe orizontală și verticală.
- Nebunul se poate muta în linie dreaptă pe orice pătrat, doar pe diagonală.
- Calul se poate muta pe cel mai apropiat pătrat care nu se află pe același rând, coloană și diagonală. Se poate înțelege drept o mișcare în „L”, adică 2 pătrate pe verticală și

unul pe orizontală sau 2 pătrate pe orizontală și unul pe verticală. Această piesă beneficiază de excepția în care mutarea ei nu poate fi blocată de o altă piesă, se consideră că „șare” la noua sa locație.

- Pionul are cele mai complexe reguli de mișcare. Un pion se poate muta un pătrat în față sau chiar două dacă este prima sa mutare, însă doar în cazul în care pătratul este liber. Aceasta poate captura o piesă inamică dacă aceasta se află pe una din cele două diagonale din față sa. Pionul nu se poate muta înapoi și mai poate face două mutări speciale: en passant și promovarea, ce urmează să fie descrise în subsecțiunea 3.1.3.

3.1.3. Mișcări speciale

În cadrul acestui joc, există și câteva mutări diferite față de cele generale:

- Rocada reprezintă o schimbare de poziții între rege și tură, mai exact, regele se mută cu 2 poziții în direcția turei, iar tura se mută în direcția opusă, adiacent față de rege. Această mutare se poate întâmpla doar sub anumite condiții: regele și tura implicate să nu se fi mutat până deloc până atunci, să nu existe piese între rege și tură, regele să nu fie atacat sau unul din pătratele prin care va trece să nu fie atacate.
- En passant se poate întâmpla atunci când un pion se mută cu două poziții în față și ajunge adiacent față de un pion inamic. Oponentul își poate muta pionul pe diagonală pe coloana pe care se află pionul ce s-a mutat inițial, capturându-l. Aceasta mutare se poate face doar imediat după ce s-a făcut mutarea inițială cu două poziții în față.
- Promovarea unui pion se întâmplă atunci când pionul ajunge pe ultimul rând al tablei. Aceasta are opțiunea să se transforme în regină, tură, nebun sau cal, indiferent de numărul de piese de pe tablă. Astfel, în teorie este posibil ca un jucător să aibă 9 regine sau chiar 10 ture, nebuni sau cai.

3.2. Aspectul jocului

Aspectul jocului se referă la reprezentarea vizuală a tablei de șah de 8 rânduri și 8 coloane, cu pătrate de culori alternante, cu un marcat corespunzător pentru marcarea rândurilor și a coloanelor, împreună cu adăugarea pieselor cu imagini sugestive.

3.2.1. Construirea tablei de șah

Pentru a construi tabla de șah, ne vom folosi de şablonane Django [\[5\]](#) pentru a genera

HTML[7] dinamic.

Din punct de vedere al părții de backend [6], avem o clasă de python [4] numită „Board” ce se ocupă de reprezentarea mai multor informații esențiale printre care se enumără și un dicționar numit „table” ce conține 8 alte dicționare corespunzătoare fiecărui rând, ce au ca scop memorarea poziției fiecărei piese de pe tablă. Inițial, dicționarul este inițializat cu piesele în pozițiile de start generale.

```
self.table = {
    '8': {'a': None, 'b': None, 'c': None, 'd': None, 'e': None, 'f': None, 'g': None, 'h': None},
    '7': {'a': None, 'b': None, 'c': None, 'd': None, 'e': None, 'f': None, 'g': None, 'h': None},
    '6': {'a': None, 'b': None, 'c': None, 'd': None, 'e': None, 'f': None, 'g': None, 'h': None},
    '5': {'a': None, 'b': None, 'c': None, 'd': None, 'e': None, 'f': None, 'g': None, 'h': None},
    '4': {'a': None, 'b': None, 'c': None, 'd': None, 'e': None, 'f': None, 'g': None, 'h': None},
    '3': {'a': None, 'b': None, 'c': None, 'd': None, 'e': None, 'f': None, 'g': None, 'h': None},
    '2': {'a': None, 'b': None, 'c': None, 'd': None, 'e': None, 'f': None, 'g': None, 'h': None},
    '1': {'a': None, 'b': None, 'c': None, 'd': None, 'e': None, 'f': None, 'g': None, 'h': None},
}
```

În continuare, această clasă conține o funcție denumită „render” ce folosește modulul „django.template.loader” [22] pentru a genera un şablon Django [5] ce va avea drept context toate informațiile clasei „Board” pentru a putea modifica dinamic conținutul HTML [7].

```
def render(self, context):
    template = loader.get_template(self.template_name)
    context['board'] = self
    html_table = template.render(context)
    return html_table
```

Din punct de vedere al părții de frontend [6], avem 3 fișiere de tip HTML [7] pentru a genera suprafața de joc: „table.html”, „table_row.html”, „table_square.html”. Primul este considerat drept şablonul Django [5] corespunzător pentru generarea tablei de șah, iar celelalte două sunt ajutătoare și reprezintă generarea particulară a unui rând și a unui pătrat. Practic, tabla de șah reprezintă o structură HTML [7] de tip tabel, în care fiecare celulă a tabelului reprezintă un pătrat.

În plus, ținem cont și de rolul utilizatorului ce vizionează jocul pentru a determina dacă trebuie să afișăm tabla inversată (pentru jucătorul negru) sau nu (pentru jucătorul alb). (Figura 3)

```
{% with light_color=board.light_color %}
{% with dark_color=board.dark_color %}
<table id="chess-table" border="1" class="chess-board" style="margin: auto">
```

```

{%
if user_role == "black" %}
<tr align="center" class="chess_board_bottom_line">
    <td></td><td>h</td><td>g</td><td>f</td><td>e</td><td>d</td><td>c</td><td>b</td><td>a</td>
</tr>
{%
include 'games/table_row.html' with row=board.table.1 row_ind=1 even=dark_color odd=light_color %}
{%
include 'games/table_row.html' with row=board.table.2 row_ind=2 even=light_color odd=dark_color %}
{%
include 'games/table_row.html' with row=board.table.3 row_ind=3 even=dark_color odd=light_color %}
{%
include 'games/table_row.html' with row=board.table.4 row_ind=4 even=light_color odd=dark_color %}
{%
include 'games/table_row.html' with row=board.table.5 row_ind=5 even=dark_color odd=light_color %}
{%
include 'games/table_row.html' with row=board.table.6 row_ind=6 even=light_color odd=dark_color %}
{%
include 'games/table_row.html' with row=board.table.7 row_ind=7 even=dark_color odd=light_color %}
{%
include 'games/table_row.html' with row=board.table.8 row_ind=8 even=light_color odd=dark_color %}
{%
else %}
{%
include 'games/table_row.html' with row=board.table.8 row_ind=8 even=light_color odd=dark_color %}
{%
include 'games/table_row.html' with row=board.table.7 row_ind=7 even=dark_color odd=light_color %}
{%
include 'games/table_row.html' with row=board.table.6 row_ind=6 even=light_color odd=dark_color %}
{%
include 'games/table_row.html' with row=board.table.5 row_ind=5 even=dark_color odd=light_color %}
{%
include 'games/table_row.html' with row=board.table.4 row_ind=4 even=light_color odd=dark_color %}
{%
include 'games/table_row.html' with row=board.table.3 row_ind=3 even=dark_color odd=light_color %}
{%
include 'games/table_row.html' with row=board.table.2 row_ind=2 even=light_color odd=dark_color %}
{%
include 'games/table_row.html' with row=board.table.1 row_ind=1 even=dark_color odd=light_color %}
<tr align="center" class="chess_board_bottom_line">
    <td></td><td>a</td><td>b</td><td>c</td><td>d</td><td>e</td><td>f</td><td>g</td><td>h</td>
</tr>
%endif %
</table>
%endwith %
%endwith %

```

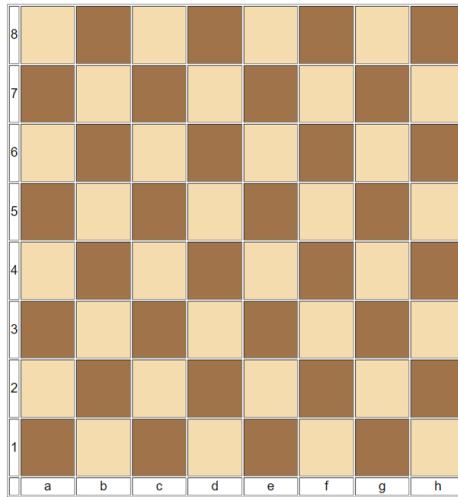


Figura 3.3 - Tabla de şah generată

3.2.2. Adăugarea pieselor cu imagini specifice

Pentru a avea un joc intuitiv și cât mai placut din punct de vedere estetic, este important să adăugăm imagini sugestive. Acest aspectul vizual este important pentru a

îmbunătăți experiența jucătorilor, oferind o imagine clară asupra tipurilor de piese de pe tablă.

În ceea ce privește dezvoltarea pe partea de backend [6], o piesă va fi reprezentată printr-o clasă specifică tipului ei. Clasele sunt organizate ierarhic și toate moștenesc dintr-o clasă de bază numită „Piece”. Aceasta are deja câteva metode implementate, cum ar fi verificarea dacă o mutare este în afara limitei tablei sau obținerea piesei de la o anumită poziție și o metodă ce trebuie implementată de fiecare clasă în parte, cea care întoarce toate mutările piesei.

```
class Piece:
    def __init__(self, color):
        self.color = color

    def getAvailableMoves(self, board, from_row, from_col):
        raise NotImplementedError("This method should be overridden by subclasses")

    def __str__(self):
        raise NotImplementedError("This method should be overridden by subclasses")

    def outOfBounds(self, row, col):
        row = int(row)
        if row < 1 or 8 < row:
            return True

        # out of bounds col
        if col < 'a' or 'h' < col:
            return True

        return False

    def getPiece(self, board, row, col):
        piece = board.get_piece(row, col)
        return piece
```

Pentru atașarea pieselor pe tablă, dicționarul „table” din clasa „Board” va fi inițializat cu căte un obiect al fiecărei clase corespunzătoare tipului de piesă ce trebuie să se afle la o anumită poziție.

```
self.table = {
    '8': {'a': Rook('black'), 'b': Knight('black'), 'c': Bishop('black'), 'd': Queen('black'), 'e': King('black'), 'f': Bishop('black'), 'g': Knight('black'), 'h': Rook('black')},
    '7': {'a': Pawn('black'), 'b': Pawn('black'), 'c': Pawn('black'), 'd': Pawn('black'), 'e': Pawn('black'), 'f': Pawn('black'), 'g': Pawn('black'), 'h': Pawn('black')},
    '6': {'a': None, 'b': None, 'c': None, 'd': None, 'e': None, 'f': None, 'g': None, 'h': None},
    '5': {'a': None, 'b': None, 'c': None, 'd': None, 'e': None, 'f': None, 'g': None, 'h': None},
    '4': {'a': None, 'b': None, 'c': None, 'd': None, 'e': None, 'f': None, 'g': None, 'h': None},
    '3': {'a': None, 'b': None, 'c': None, 'd': None, 'e': None, 'f': None, 'g': None, 'h': None},
    '2': {'a': Pawn('white'), 'b': Pawn('white'), 'c': Pawn('white'), 'd': Pawn('white'), 'e': Pawn('white'), 'f':
```

```

Pawn('white'), 'g': Pawn('white'), 'h': Pawn('white')},
  '1': {'a': Rook('white'), 'b': Knight('white'), 'c': Bishop('white'), 'd': Queen('white'), 'e': King('white'), 'f':
Bishop('white'), 'g': Knight('white'), 'h': Rook('white')},
}

```

Pe partea de frontend, am ales drept imagini setul popular folosit de majoritatea utilizatorilor de pe Lichess [2], valabil pe github-ul destinat jocului de șah de pe această platformă. [23] (Figura 4)



Figura 3.4 - Imaginele pieselor de șah

În ceea ce privește atașarea pieselor, în funcție de informația din celula din tabela HTML [7], imaginea corespunzătoare este încărcată corespunzător și afișată în elementul „td”.

```

<td bgColor="#{{ bgcolor }}" data-position="{{ r_ind }}{{ s_ind }}" class="chess-square">
  {% with folder="/static/chess/pieces/" %}
  {% if square %}
    
  {% endif %}
  {% endwith %}
</td>

```

Având în vedere toate detaliile menționate, aşa arată varianta finală a tablei de șah:

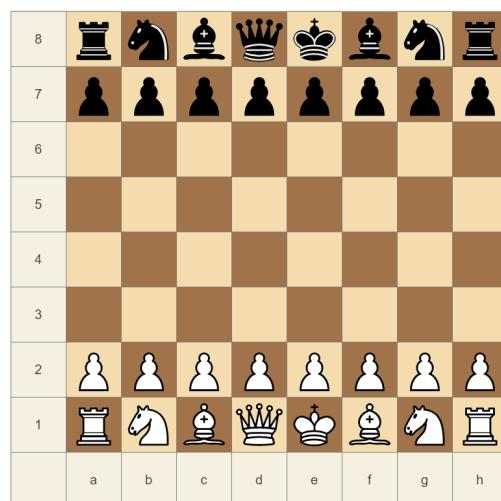


Figura 3.5 - Tabla de șah generată cu piese

3.3. Funcționalități principale

În secțiunea aceasta, vom discuta despre principalele funcționalități ale jocului nostru. Asta implică implementarea mutărilor, buna derulare a unui meci de șah și câteva alte elemente finale.

3.3.1. Acțiunea de a muta o piesă

Pentru început, trebuie să dezvoltăm acțiunea de a muta o piesă, care este cea mai importantă mecanică a jocului. Aceasta se va realiza prin 2 modalități diferite: click sau „drag and drop”.

În acest sens, vom avea nevoie să introducem un nou fișier de tip JavaScript [9] numit „chess_interactions.js” cu scopul de a permite jucătorului să interacționeze cu piesele. Acesta va fi atașat ca un script asupra şablonului Django [5] responsabil de vizualizarea paginii ce conține jocul de șah.

```
<script src="{% static 'js/chess_interaction.js' %}"></script>
```

Modul său de funcționare se rezumă la adăugarea unor metode numite „event listener” [24] ce vor fi de tipuri diferite în funcție de faptul dacă pătratele conțin piese sau nu. Asfel, trebuie ca mai întâi să selectăm piesele și zonele libere în două variabile diferite. Asta se realizează prin apelarea funcției „querySelectorAll” [25], ce alege toate elementele HTML [7] ce conțin un anumit selector.

```
const pieces = document.querySelectorAll("img[draggable='true']");
const squares = document.querySelectorAll("td[data-position]");
```

Pentru elementele ce conțin piese de joc, vom adăuga un „event listener” [24] de tip „drag start” și unul de tip „click”, corespunzătoare celor 2 moduri diferite prin care se poate face o mutare. Pentru zonele libere, vom adăuga un „event listener” [24] de tip „dragover” și unul de tip „drop” ce vor aștepta ca o piesă să fie trasă și așezată în acea poziție.

```
pieces.forEach(piece => {
    const pieceColor = piece.getAttribute("data-color");
    if (pieceColor === userRole) {
```

```

        piece.addEventListener("dragstart", dragStart);
        piece.addEventListener("click", handleClick);
        piece.setAttribute("draggable", true);
    }
});

squares.forEach(square => {
    square.addEventListener("dragover", dragOver);
    square.addEventListener("drop", drop);
});

```

În cazul în care se detectează un anumit eveniment, se apelează funcții ce se ocupă cu transferul de informație a elementelor HTML [7] ce trebuie să se execute pentru a înfățișa corect schimbarea poziției piesei. Iar pentru a ne asigura că pozițiile și mutările sunt sincronizate și în backend, vom face un request de tip POST [18] la endpoint-ul „/move_piece/{game_id}”, unde „game_id” reprezintă identificatorul unic al jocului, ce va conține în body coordonatele mutării și, dacă răspunsul primit înapoi de la server este în regulă, atunci mutarea se va face cu succes. Scopul acestui procedeu este de a valida mutările jucătorilor și pentru a preveni posibilitatea unor mutări ilegale.

```

function makeMove(fromPosition, toPosition, targetSquare, promotion = null) {
    fetch(`/move_piece/${gameId}`, {
        method: "POST",
        headers: {
            "Content-Type": "application/json",
            "X-CSRFToken": getCookie("csrftoken"),
        },
        body: JSON.stringify({ from: fromPosition, to: toPosition, turn: turn, promotion: promotion })
    })
    .then(response => response.json())
    .then(data => {
        if (data.status === "ok") {
            if (data.enPassant) {
                const fromRow = fromPosition[0];
                const capturedPosition = fromRow + toPosition[1];
                resetSquare(capturedPosition);
            }
            resetSquare(fromPosition);
            resetSquare(toPosition);
            if (targetSquare) {
                targetSquare.appendChild(selectedPiece);
            }
            turn = data.new_turn;
            document.getElementById("turn").value = turn;
            clearHighlights();
            updateDraggable();
            if (promotion) {
                promotePawn(toPosition, userRole[0] + promotion); // e.g., "wQ" for white Queen
            }
        }
    })
}

```

```
}
```

Pentru a îmbunătăți experiența utilizatorului, în cadrul mutării piesei prin click, vom recomanda jucătorului toate mutările posibile pe care le poate face cu acea piesă. Această funcționalitate se va implementa prin apelarea unui request de tip GET [\[18\]](#) la endpoint-ul „/get_moves/{game_id}?from={from_position}&turn={turn}” în care „game_id” semnifică identificatorul unic al meciului, „from” reprezintă coordonatele piesei, iar „turn” este jucătorul care face mutarea. Răspunsul primit de la acest request va conține o listă de mutări pe care le vom marca cu o bulină verde pentru pozițiile libere sau cu un fundal roșu transparent pentru locurile unde se află piese inamice, prin intermediul funcțiilor „clearHighlights” și „highlightMoves”.

```
function handleClick(e) {
  const piece = e.target;
  const fromPosition = piece.parentNode.getAttribute("data-position");
  if (selectedPiece === piece) {
    clearHighlights();
    selectedPiece = null;
    return;
  }
  clearHighlights();
  selectedPiece = piece;
  fetch(`/get_moves/${gameId}?from=${fromPosition}&turn=${turn}`, {
    method: "GET",
    headers: {
      "Content-Type": "application/json"
    }
  })
  .then(response => response.json())
  .then(data => {
    if (data.status === "ok") {
      highlightMoves(data.moves, fromPosition);
    } else {
      console.error("Failed to get moves");
    }
  });
}
```

Momentan, nu avem validare pe mutări, deci se vor evidenția toate pozițiile.

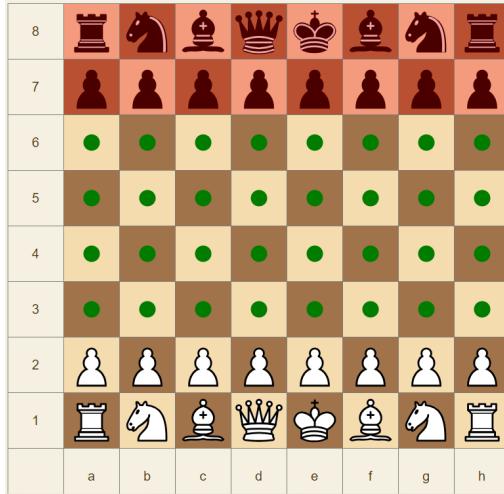


Figura 3.6 - Evidențiere poziții valide

3.3.2. Mutarea corectă a pieselor

Mutarea corectă a pieselor de joc reprezintă o operație complexă ce necesită implementarea corectă a direcțiilor în care o piesă își poate schimba direcția și implică o verificare a mutărilor în care ne asigurăm că orice schimbare a poziției unei piese nu îl lasă pe propriul rege în „șah”.

În această secțiune, vom detalia modul în care backend-ul [6] răspunde apelului „/move_piece/{game_id}” de la frontend [6] și vom explica cum sunt implementate mutările specifice fiecărui tip de piesă. La fiecare apel al acestui request, se creează un nou obiect de tip „Play”, responsabil pentru a instanția clasa Board cu toate informațiile specifice poziției în care se află meciul de șah. Acesta apelează metoda „getSafeMoves” din clasa „Piece”, ce returnează toate mutările legale ale piesei respective (prin „getAvailableMoves”), filtrate după verificarea că oricare din aceste mutări să nu lase propriul rege descoperit în „șah” (prin „is_valid_move”). Funcția „getAvailableMoves” este o metodă implementată specific de toate clasele corespunzătoare fiecărei piese în parte ce returnează toate mutările posibile.

```
def getSafeMoves(self, board, from_row, from_col):
    all_moves, enPassantPos, castling = self.getAvailableMoves(board, from_row, from_col)
    safe_moves = []
    for move in all_moves:
        to_row, to_col = move[0], move[1]
        if board.is_valid_move(from_row, from_col, to_row, to_col):
            safe_moves.append(move)
    return safe_moves, enPassantPos, castling
```

Implementarea mutărilor generale ale unui pion implică mutarea înainte cu una sau două poziții, asigurarea că nu există o piesă ce poate să blocheze mișcarea și mutarea pe diagonală pentru capturarea unei piese inamice.



Figura 3.7 - Mutarea pionului înainte



Figura 3.8 - Capturarea cu pionul

Mutările calului se rezumă la o mișcare în „L”, acesta având posibilitatea să sară peste piese și să nu fie blocat.



Figura 3.9 - Mutările calului

Nebunul se mută pe diagonale și poate continua să se ducă în direcția aleasă până când întâlnește o piesă. Implementarea presupune verificarea posibilității de a muta până când o piesă blochează mișcarea sau până când nebunul ar ieși de pe tablă.



Figura 3.10 - Mutările nebunului

Tura se mută pe verticală sau orizontală și poate continua să se ducă în direcția aleasă până când întâlnește o piesă. Implementarea presupune verificarea că tura nu ar ieși de pe tablă și dacă o piesă poate bloca mișcarea.

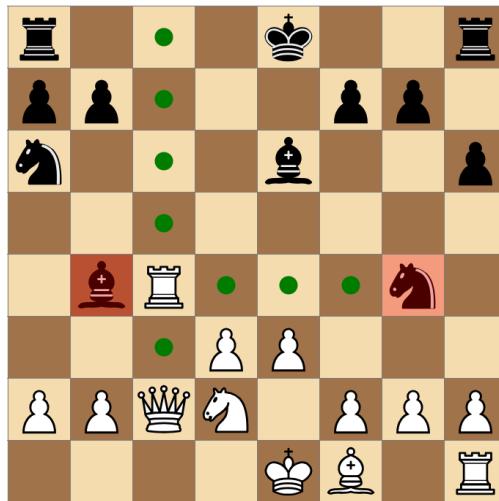


Figura 3.11 - Mutările turei

Regina mută în toate direcțiile: verticală, orizontală și diagonală. Implementarea mutărilor sale se rezumă doar la a prelua logica dezvoltată pentru mutările turei și a nebunului.

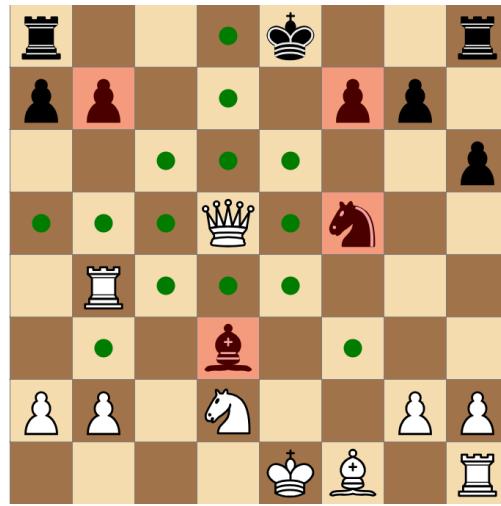


Figura 3.12 - Mutările reginei

Regele se poate muta în toate direcțiile, însă poate înainta doar cu o singură poziție. În plus, acesta nu se poate muta la distanță de un singur pătrat de regele inamic sau într-un loc aflat sub control inamic.

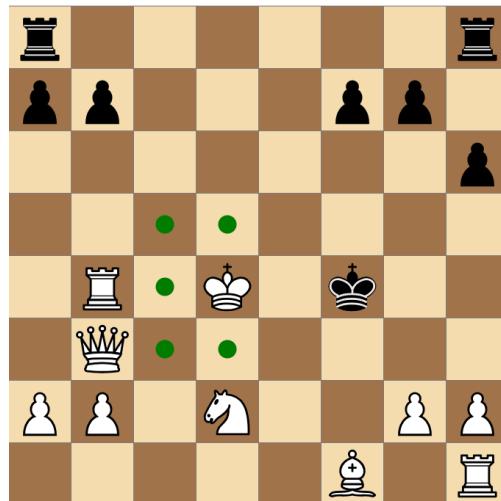


Figura 3.13 - Mutările regelui

3.3.3. Situații speciale

Rocada [10] reprezintă mutarea în care tura și regele fac schimb de poziții, cu scopul de a proteja regele. Aceasta presupune mutarea regelui cu 2 poziții, în loc de una, iar tura ocupând poziția vecină regelui din direcția din care acesta vine. Din punct de vedere al implementării, pe partea de backend, vom calcula posibilitatea efectuării acestei mutări, prin verificarea dacă o piesă inamică a pus regele în „șah” sau dacă atacă oricare din pozițiile prin

care trece regele și dacă piesele implicate în rocadă au mai fost mutate până acum. În plus, când se trimite răspunsul înapoi la frontend, se va adăuga încă un parametru în plus ce ar conține valoarea „Q” sau „K”, indicând direcția în care regele vrea să facă rocadă (în direcția reginei sau a regelui). Astfel, când acest parametru este primit în frontend, ne asigurăm că schimbarea de piese se întâmplă în mod corect.



Figura 3.14 - Rocada disponibilă



Figura 3.15 - Rocada blocată

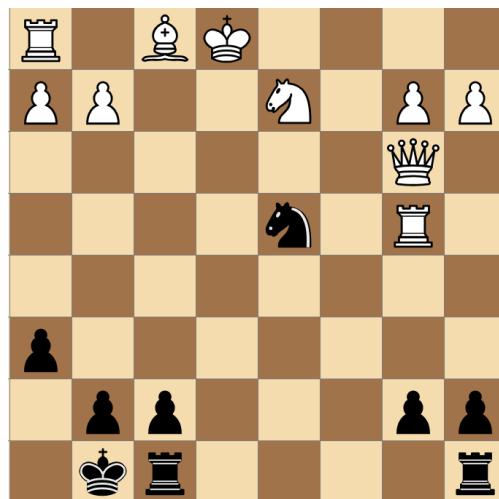


Figura 3.16 - Așezarea pieselor după rocadă

Mutarea „en passant” [10] reprezintă una din mutările speciale ale pionului, în care acesta poate capture un pion inamic care abia ce s-a deplasat cu două poziții în față și se află adiacent față de acesta. Implementarea presupune verificarea pozițiilor corespunzătoare ale pionilor, împreună cu asigurarea că pionul ce urmează să fie capturat a fost cel care a mutat ultimul și transmiterea unui alt parametru în răspunsul trimis înapoi către frontend pentru a

face schimbarea de poziții corect.



Figura 3.17 - En Passant - Înainte



Figura 3.18 - En Passant - După

Promovarea [10] este a doua mutare specială a pionului. Aceasta se întâmplă atunci când pionul ajunge pe ultimul rând și se transformă în alta piesă (nebun, cal, tură, regină), la alegerea jucătorului. Implementarea presupune detectarea acestui tip de mutare, atât în backend cât și în frontend, pentru a putea actualiza starea tablei corespunzător în funcție de ce piesă alege utilizatorul.



Figura 3.19 - Promovare - Înainte

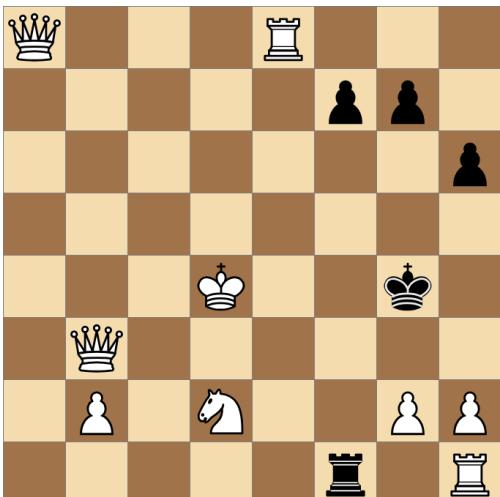


Figura 3.20 - Promovare - După

3.3.4. Actualizare mutări în timp real

Actualizarea mutărilor în timp real este esențială pentru aplicația noastră deoarece cuprinde un joc multiplayer. Fiecare jucător trebuie să vadă mutările adversarului imediat ce acestea sunt făcute, fără a restara pagina. Pentru a realiza această funcționalitate, vom folosi Django Channels [26].

Channels este un proiect care face parte din framework-ul Django [5] ce facilitează comunicarea bidirectională în timp real. Acesta extinde capabilitățile framework-ului Django [5] pentru a suporta protocoale de comunicare web asincrone precum WebSockets [27]. În loc să folosească HTTP pentru comunicarea unidirectională tradițională, WebSockets [27] permit o conexiune persistentă între client și server, permitând serverului să trimită date către client în timp real.

În acest sens, fiecare partidă de șah va avea un WebSocket [27] dedicat la care cei 2 jucători, dar și alți utilizatori spectatori, se vor putea conecta pentru a vedea mutările în timp real.

Pentru a realiza asta, va trebui să configuriăm aplicația să funcționeze asincron (ASGI, în loc de WSGI) [28]. În „settings.py”, adăugăm configurația necesară pentru asta și pentru Django Channel Layers [26].

```
ASGI_APPLICATION = 'uni_chess.asgi.application'
CHANNEL_LAYERS = {
    'default': {
        'BACKEND': 'channels.layers.InMemoryChannelLayer',
    }
}
```

Mai departe, creăm un nou fișier „asgi.py” pentru a defini aplicația ASGI [28]. „ProtocolTypeRouter” este folosit pentru a gestiona atât request-uri HTTP cât și WebSockets [27].

```
import django
from django.core.asgi import get_asgi_application
from channels.routing import ProtocolTypeRouter, URLRouter
from channels.auth import AuthMiddlewareStack
from games import routing

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'uni_chess.settings')
django.setup()
application = ProtocolTypeRouter({
    "http": get_asgi_application(),
    "websocket": AuthMiddlewareStack(
```

```
URLRouter(  
    routing.websocket_urlpatterns ) ), } )
```

Pentru a defini corect ruta URL a WebSocket-ului [27] unui joc, va trebui să adăugăm un fișier „routing.py” care să mapeze ruta cu „consumerul” conexiunii noastre.

```
from django.urls import re_path  
from . import consumers  
  
websocket_urlpatterns = [  
    re_path(r'^ws/game/(?P<game_id>\d+)/$', consumers.GameConsumer.as_asgi()),  
]
```

Un „consumer” reprezinta o clasă ce gestioneaza viața unui WebSocket [27] și informațiile acestuia. În cazul nostru, vom crea un nou fișier numit „consumers.py” ce va conține clasa „GameConsumer”.

```
import json  
from channels.generic.websocket import AsyncWebsocketConsumer  
class GameConsumer(AsyncWebsocketConsumer):  
    async def connect(self):  
        self.game_id = self.scope['url_route']['kwargs']['game_id']  
        self.game_group_name = f'game_{self.game_id}'  
        await self.channel_layer.group_add(  
            self.game_group_name,  
            self.channel_name  
        )  
        await self.accept()  
    async def disconnect(self, close_code):  
        await self.channel_layer.group_discard(  
            self.game_group_name,  
            self.channel_name  
        )  
        await self.channel_layer.group_send(  
            self.game_group_name,  
            {  
                'type': 'game_move',  
                'from': from_pos,  
                'to': to_pos,  
                'turn': turn,  
                'EnPassant': enPassant,  
                'checkmate': checkmate,  
                'promotion': promotion,  
                'castling': castling  
            }  
        )  
    async def game_move(self, event):  
        print('Send: ', event)  
        from_pos = event['from']  
        to_pos = event['to']  
        turn = event['turn']
```

```

enPassant = event.get('enPassant', False)
checkmate = event.get('checkmate', False)
promotion = event.get('promotion', False)
castling = event.get('castling', False)
await self.send(text_data=json.dumps({
    'from': from_pos,
    'to': to_pos,
    'turn': turn,
    'enPassant': enPassant,
    'checkmate': checkmate,
    'promotion': promotion,
    'castling': castling,
}))
```

În ceea ce privește partea de frontend, în cadrul scriptului „chess_interactions.js” vom deschide o conexiune WebSocket [27] către server, utilizând URL-ul specific unui joc și vom asculta mesajele primite, actualizând interfața jucătorilor în funcție de datele primite.

```

const socket = new WebSocket(`ws://${window.location.host}/ws/game/${gameId}`);
socket.onmessage = function(e) {
    const data = JSON.parse(e.data);
    const from = data.from;

    const to = data.to;
    turn = data.turn;
    const enPassant = data.enPassant || false;
    document.getElementById("turn").value = turn;
    document.getElementById("turn-display").innerText = turn;
    const piece = document.querySelector(`td[data-position="${from}"] img`);
    const targetSquare = document.querySelector(`td[data-position="${to}"]`);
    if (enPassant) {
        const fromRow = from[0];
        const capturedPosition = fromRow + to[1];
        resetSquare(capturedPosition);
    }
    if (data.castling) {
        performCastlingMove(data.castling, from);
    } else {
        resetSquare(from);
        resetSquare(to);
        targetSquare.appendChild(piece);
    }
    updateDraggable();
    if (data.promotion) {
        const promotionColor = turn === "white" ? "b" : "w"
        promotePawn(to, promotionColor + data.promotion);
    }
    if (data.checkmate) {
        displayCheckmateMessage(turn);
    }
}
```

```
    }  
};
```

3.3.5. Administrarea timpului

Administrarea timpului reprezintă un aspect important al jocului de șah. Partidele se pot juca în diferite formate de timp: Blitz (1-5 minute), Rapid(5-30 minute) sau Classic (de la 30 de minute până la nelimitat) . De fiecare dată când un jucător este la mutare, acestuia î se scade din timp până când va face mutare, caz în care timpul său î se oprește și , dacă formatul de timp are și increment, atunci timpul poate să î mai crească cu câteva secunde. Dacă timpul unui jucător expiră, atunci jucătorul a pierdut automat partida.

Implementarea acestei funcționalități implică mai multe componente ce lucrează împreună pentru a asigura trecerea corectă a timpului. Pentru început, vom adăuga 2 noi proprietăți modelului „Game” pentru a ține cont de cât timp mai are fiecare jucător, acestea vor fi actualizate după fiecare mutare a unui jucător pentru a menține aceste informații persistente în partea de backend [6].

Pentru a dezvolta logica de cronometrare a timpului, vom adăuga un nou fișier numit „timer.js”, responsabil administrarea timpului jucătorilor. Astfel, atunci când pagina de joc se încarcă, scriptul va inițializa cronometrele și le va păstra valorile în „sessionStorage” [29] pentru a le menține persistente în timp real, chiar și după se reîncarcă pagina.

```
function initializeTimers() {  
    whiteTime = parseInt(window.sessionStorage.getItem(COUNTER_KEY_WHITE)) ||  
    parseInt(document.getElementById("initial_white_time").value) * 60;  
    blackTime = parseInt(window.sessionStorage.getItem(COUNTER_KEY_BLACK)) ||  
    parseInt(document.getElementById("initial_black_time").value) * 60;  
  
    whiteTimerElement.textContent = formatTime(whiteTime);  
    blackTimerElement.textContent = formatTime(blackTime);  
  
    let turn = document.getElementById("turn").value;  
    startTimer(turn);  
}
```

După aceea, acestea vor fi actualizate corespunzător, scazând cu câte o secundă pentru cel care trebuie să facă mutarea și rămânând la fel pentru cel care nu trebuie să mute. Atunci când un jucător mută o piesă, scriptul „chess_interactions.js” va trimite împreună și valorile din sessionStorage” pentru a actualiza în backend proprietățile modelului „Game”.

```
body: JSON.stringify({  
    from: fromPosition,
```

```
    to: toPosition,  
    turn: turn,  
    promotion: promotion,  
    white_time_remaining: turn === 'white' ? whiteTime + increment : whiteTime,  
    black_time_remaining: turn === 'black' ? blackTime + increment : blackTime  
  },  
);
```

Pentru a avea cronometrele sincronizate în ambele părți, după ce o mișcare este validată și procesată în backend, valorile timpilor rămași vor fi trimise prin WebSocket [27]. Metoda „socket.onmessage” din „timer.js” se va ocupa de actualizarea corectă a cronometrelor.

```
socket.onmessage = function(e) {
    const data = JSON.parse(e.data);
    let turn = data.turn;
    whiteTime = data.white_time_remaining;
    blackTime = data.black_time_remaining;
    whiteTimerElement.textContent = formatTime(whiteTime);
    blackTimerElement.textContent = formatTime(blackTime);
    startTimer(data.turn);
};
```

Astfel, după câteva stilizări, aşa arată pagina cu tabla de şah, acum şi cu căsutele pentru cronometre.



Figura 3.21 - Tablă de sah după adăugare timp

3.4. Elemente finale

În această secțiune, vom explica ultimele funcționalități adăugate jocului de sah: butonul de renunțare, butonul de remiză și detectarea finalului de meci prin „pat”.

În ceea ce privește logica de renunțare, vom adăuga un buton pe care fiecare jucător va avea posibilitatea să îl apese atunci când acesta vrea să renunțe la partidă și să anunțe că s-a lăsat învins. Pentru a realiza asta, vom adăuga un buton între cele 2 cronometre ale jucătorilor. Acesta va avea atașat un „event listener” [24] ce va asculta dacă cineva va face click pe el. În momentul apasării, utilizatorul va primi un mesaj de tip „alert” pentru confirmarea renunțării. După confirmare, se va trimite un request către server la endpoint-ul „/resign/{gameId}” pentru a schimba starea jocului și pentru a încheia partida.

```
document.getElementById("resign-button").addEventListener("click", function() {
  if (confirm("Are you sure you want to resign?")) {
    fetch('/resign/${gameId}', {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
        "X-CSRFToken": getCookie("csrftoken")
      }
    })
    .then(response => response.json())
    .then(data => {
      if (data.status === "ok") {
        const message = `${data.loser} resigns! ${data.winner} wins!`;
        displayEndgameMessage(message);

      } else {
        console.error("Failed to resign");
      }
    });
  }
});
```

Butonul de a oferi remiză permite unui jucător să întrebe adversarul dacă este de acord ca meciul să se termine într-o remiză. Din punct de vedere al implementării, vom adăuga un alt buton lângă cel de renunțare care va avea 3 texte diferite: „Offer Draw”, „Cancel Draw” și „Accept Draw”. Inițial, ambii jucători vor avea posibilitatea de a oferi remiză. Dacă unul

dintre ei apasă butonul, atunci acesta își va schimba textul în „Cancel Draw”, iar adversarului în „Accept Draw”. Dacă se apasă butonul de anulare, atunci textul va reveni înapoi pentru ambii utilizatori, iar dacă oponentul acceptă remiza, atunci meciul se încheie prin egalitate. Pentru a realiza asta, vom avea un „event listener” [24] pentru acest buton ce va face un request diferit la server în funcție de mesajul pe care acesta îl conține. Serverul va avea 3 endpoint-uri deschise „,/offer_draw/{gameId}”, „,/cancel_draw/{gameId}” și „,/accept_draw/{gameId}”.

Pentru a avea sincronizare în timp real între jucători, serverul va trimite la WebSocket [27] un mesaj de tip corespunzător cererii făcute, iar din frontend vom asculta după asemenea mesaje și vom acționa în funcție de ele.

```
document.getElementById("offer-draw-button").addEventListener("click", function() {
  const offerDrawButton = document.getElementById("offer-draw-button");
  if (offerDrawButton.textContent === "Offer Draw") {
    fetch(`/offer_draw/${gameId}`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
        "X-CSRFToken": getCookie("csrftoken")
      },
      body: JSON.stringify ({
        turn: userRole
      })
    })
    .then(response => response.json())
    .then(data => {
      if (data.status === "ok") {
        offerDrawButton.textContent = "Cancel Draw";
      } else {
        console.error("Failed to offer draw");
      }
    });
  } else if (offerDrawButton.textContent === "Cancel Draw"){
    fetch(`/cancel_draw/${gameId}`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
        "X-CSRFToken": getCookie("csrftoken")
      }
    })
    .then(response => response.json())
    .then(data => {
      if (data.status === "ok") {
        offerDrawButton.textContent = "Offer Draw";
      } else {
        console.error("Failed to cancel draw offer");
      }
    });
  }
});
```

```
        }
    });
}
else {
    fetch('/accept_draw/${gameId}', {
        method: "POST",
        headers: {
            "Content-Type": "application/json",
            "X-CSRFToken": getCookie("csrftoken")
        }
    })
    .then(response => response.json())
    .then(data => {
        if (data.status === "ok") {
            handleDrawResponse();
        } else {
            console.error("Failed to cancel draw offer");
        }
    });
}
});
```

Nu în ultimul rând, vom adăuga și funcționalitatea de detectare a finalizării partidei prin „pat”. Acest eveniment se întâmplă atunci când unul dintre jucători rămâne fără mutări posibile și nici nu se află în „șah”. Din punct de vedere al implementării, atunci când verificăm toate mutările posibile ale unui jucător, dacă acesta nu mai are nicio mutare disponibilă și nici nu atacă regele inamic, atunci vom încheia meciul forțat prin remiză.

Capitolul 4

Analiza unei partide

4.1. Introducere

În acest capitol, vom descrie procesul de implementare a funcționalității de analiză a partidelor de șah. Vom parcurge toate etapele necesare pentru a dezvolta această unealtă pentru a-i oferi utilizatorului un mijloc de creștere a nivelului său de joc.

În lumea modernă a șahului, deoarece modelele de AI [11] s-au dezvoltat enorm, jucătorul profesionist de șah trebuie să își analizeze meciurile folosind un program mult mai avansat decât orice om, pentru a ajunge la cel mai înalt nivel. Astfel, integrând un motor puternic, putem evalua mai bine o poziție de șah, putem identifica greșeli mai ușor și putem descopri strategii noi.

4.1.1. Descrierea funcționalității

Funcționalitatea pe care o introducem constă într-un sistem de analiză interactivă a unei partide de șah terminate. Atunci când un utilizator vizitează o pagină ce conține un meci finalizat, acesta va avea posibilitatea să apese pe un buton ce îl va redirecționa către pagina dedicată analizei unui joc. Aici, utilizatorului i se va afișa tabla de șah cu piesele în pozițiile inițiale și va putea să navigheze prin fiecare mutare a partidei, apăsând pe două săgeți. De fiecare dată când va naviga prin mutări, acesta va primi sugestii de la modelul AI [11] cu cele mai bune mișcări pe care le poate face, precum și evaluarea poziției jocului. În cele din urmă, ne vom asigura că funcționalitatea are un aspect vizual plăcut prin punerea în evidență a mutărilor recomandate.

4.2. Implementarea paginii

În această secțiune, vom explora în detaliu implementarea paginii de analiză. Vom prezenta structura paginii, mecanismul de navigare prin mutări și modul de evidențiere a mutării curente.

4.2.1. Structura paginii

Pentru a avea o interfață plăcută pentru utilizator, vom structura pagina în 3 secțiuni principale:

- **Evaluare și sugestii:** această secțiune se află în partea stângă a paginii și afișează informațiile furnizate de modelul AI [11]: evaluarea curentă a stării jocului și o listă cu cele mai bune mutări recomandate pentru poziția curentă.
- **Tabla de șah:** aceasta este partea centrală a paginii, unde se vede tabla de șah ce este generată astfel încât să putem naviga mutare cu mutare și să observăm mutările recomandate prin intermediul unor săgeți ajutătoare.
- **Mutările partidei:** această secțiune se află în partea dreapta a paginii și ilustrează lista cu mutările partidei, iar mutarea curentă este evidențiată corespunzător pentru a fi mai ușor ca utilizatorul să urmărească mutările.

4.2.2. Navigarea prin mutări

Pentru început, vom crea un nou fișier javascript numit „analyse_game.js” ce va rula ca un script peste template-ul dedicat paginii de analiză. Pentru a permite utilizatorilor să navigheze prin mutările partidei, am implementat două butoane de navigare (înapoi și înainte). Aceste butoane vor apela funcții ce vor incrementa sau decrementa indexul mutării curente și vor actualiza tabla de șah pentru a afișa noua mutare.

Funcția „navigateMoves” schimbă indexul mutării curente și apelează funcția „updateBoard” pentru a actualiza tabla de șah.

```
function navigateMoves(direction) {
    currentMoveIndex += direction;
    if (currentMoveIndex < 0) { currentMoveIndex = 0; }
    else if (currentMoveIndex > moves.length) {
```

```

        currentMoveIndex = moves.length;
    }
    updateBoard(currentMoveIndex);
}
function updateBoard(index) {
    fetch(`/analyse/${gameId}/move`, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
            'X-CSRFToken': getCookie('csrftoken')
        },
        body: JSON.stringify({ indice: index })
    })
    .then(response => response.json())
    .then(data => {
        if (data.status === "ok") {
            highlightCurrentMove(index);
            const jsonTable = JSON.parse(data.json_table);
            renderBoard(jsonTable);
            updateEvaluation(data.evaluation);
            updateSuggestions(data.suggestions);
        } else {
            console.error("Failed to update board");
        }
    })
    .catch(error => console.error("Error:", error));
}

function renderBoard(jsonTable) {
    for (const [row, columns] of Object.entries(jsonTable)) {
        for (const [col, piece] of Object.entries(columns)) {
            const position = `${row}${col}`;
            const square = document.querySelector(`td[data-position="${position}"]`);
            if (square) {
                square.innerHTML = piece !== 'None' ? `![${piece}](/static/chess/pieces/${piece}.svg)` : '';
            }
        }
    }
}

```

4.2.3. Evidențierea mutării curente

Pentru a îmbunătăți vizibilitatea mutării curente, am implementat o funcționalitate de evidențiere. Atunci când utilizatorul navighează prin mutări, celula corespunzătoare mutării curente din lista de mutări va fi pusă în evidență cu ajutorul unei clase de CSS [8] ce doar va schimba culoarea textului și a fundalului.

```

function highlightCurrentMove(index) {
  document.querySelectorAll('.cell-highlight').forEach(el => el.classList.remove('cell-highlight'));
  const moveRow = Math.floor(index / 2);
  const moveCol = index % 2 === 0 ? 'white' : 'black';
  const cell = document.querySelector(`.moves-table tbody tr:nth-child(${moveRow + 1}) td:nth-child(${moveCol === 'white' ? 2 : 3})`);
  if (cell) {
    cell.classList.add('cell-highlight');
  }
}

```

4.3. Integrare model AI

În această secțiune, vom oferi detalii despre procesul de integrare al unui model AI pentru a oferi evaluări și sugestii în cadrul uneltei noastre de analiză a unei partide de șah.

Stockfish [30] este numele modelului nostru integrat și reprezintă una dintre ele mai puternice tehnologii dezvoltate pentru jocul de șah. Este un proiect open-source și este folosit pe scară largă pentru analiza partidelor datorită performanței sale excepționale și a flexibilității în configurare. În contextul aplicației noastre, vom utiliza Stockfish pentru a evalua pozițiile de pe tablă și pentru a recomanda cele mai bune mutări.

4.3.1. Interacțiunea cu modelul

Pentru început, vom instala programul de pe site-ul oficial [31] pentru a avea acces local la toate funcționalitățile sale. Interacțiunea cu acest model este gestionată în backend [6] unde vom folosi pachetul „python-chess” [32] pentru a comunica cu motorul și pentru a obține evaluarea poziției curente și mutările recomandate.

```

def get_evaluation(moves):
    STOCKFISH_PATH = '../uni_chess/stockfish/stockfish-windows-x86-64-avx2.exe'
    board = chess.Board()
    for move in moves:
        uci_move = convert_to_uci(move)
        board.push_uci(uci_move)

    with chess.engine.SimpleEngine.popen_uci(STOCKFISH_PATH) as engine:
        info = engine.analyse(board, chess.engine.Limit(time=0.5), multipv=7)
        scores = []
        for i in range(min(7, len(info))):
            if info[i]['score'].is_mate():
                if info[i].get('pv'):
                    score = 'Mate in ' + str(info[i]['score'].relative.mate())

```

```

        else:
            score = 'Checkmate'
        else:
            score = str(info[i]['score'].relative.score() / 100)
            scores.append(score)

    suggestions = [(info[i]['pv'][0].uci(), scores[i]) for i in range(min(7, len(info)))]
    engine.quit()
    return scores[0], suggestions

```

În această funcție, setăm calea către executabilul Stockfish [31] și inițializăm o tablă de șah prin pachetul ajutător. Parcurgem lista mutărilor și le aplicăm pe tablă, după care folosim motorul de șah pentru a evalua poziția și pentru a genera până la 7 variante posibile. Funcția returnează evaluarea curentă și lista de sugestii cu scorurile corespunzătoare.

4.3.2. Afisarea sugestiilor

Pentru a afișa sugestiile pentru fiecare mutare în parte, am dezvoltat un proces simplu de comunicare între backend și frontend. Astfel, de fiecare dată când apăsăm o săgeată pentru a naviga cu o mutare înainte sau înapoi, frontend va trimite către backend o cerere de tip POST] [18] la endpoint-ul „analyse/{game_id}/move” ce va apela metoda „analyse_game_move” din fișierul „views.py”. Această funcție preia identificatorul jocului și mutările efectuate până în acel moment și apelează metoda „get_evaluation” pentru a putea trimite mai departe evaluarea și recomandările sub formă de răspuns JSON [33].

```

@csrf_exempt
@login_required
def analyse_game_move(request, game_id):
    game = get_object_or_404(Game, pk=game_id)

    if request.method == "POST":
        data = json.loads(request.body)
        indice = data.get("indice")
        play = Play(game.data, ind=indice)
        json_table = json.dumps(play.board.get_json_table())
        moves = game.data.split(' ')[indice]
        evaluation, suggestions = get_evaluation(moves)

        return JsonResponse({"status": "ok", "json_table": json_table, "evaluation": evaluation,
        "suggestions": suggestions})
    return JsonResponse({"status": "fail"})

```

În frontend, fișierul „analyse_game.js” se ocupă de preluarea datelor din backend și actualizează interfața utilizatorului prin apelarea funcțiilor „updateEvaluation” și „UpdateSuggestions” pentru a actualiza evaluarea și recomandările făcute.

```
function updateEvaluation(evaluation) {
  const evaluationBox = document.getElementById("evaluation-box");
  evaluationBox.textContent = `Evaluation: ${evaluation}`;
}

function updateSuggestions(suggestions) {
  const suggestionsTable =
document.getElementById("suggestions-table").getElementsByTagName("tbody")[0];
  suggestionsTable.innerHTML = "";
  for (let i = 0; i < suggestions.length; i++) {
    const row = suggestionsTable.insertRow();
    const moveCell = row.insertCell(0);
    const scoreCell = row.insertCell(1);
    moveCell.textContent = suggestions[i][0];
    scoreCell.textContent = suggestions[i][1];
  }
  drawSuggestionArrows(suggestions.slice(0, 3));
}
```

În cele din urmă, aşa arată pagina de analiză a unei partide de șah:



Figura 4.1 - Analiza unei partide de șah

Capitolul 5

Concluzie

Obiectivul nostru de a oferi o platformă online, unde studenții pot juca șah împreună, a fost atins. Acest proiect a acoperit o gamă largă de aspecte tehnice, printre care se enumeră: design-ul plăcut al interfeței, procesarea cererilor și implementarea logicii jocului de șah, gestionarea datelor în cadrul unei baze de date funcționale și integrarea unui model AI [\[11\]](#) pentru analiza unei partide de șah.

În concluzie, dezvoltarea unei aplicații de șah online reprezintă o realizare semnificativă ce pune în evidență cunoștințele dobândite pe parcursul acestor 3 ani de facultate.

Bibliografie

- [1] Adam Volle, *Web App*, <https://www.britannica.com/topic/Web-application>, Data accesării: 9 iunie 2024.
- [2] Lichess, <https://lichess.org>, Data accesării: 3 iunie 2024.
- [3] Chess.com, <https://www.chess.com/about>, Data accesării: 3 iunie 2024.
- [4] The Python Software Foundation, *Python*, <https://www.python.org/doc/essays/blurb/>, Data accesării: 5 iunie 2024.
- [5] Django, <https://www.djangoproject.com/>, Data accesării: 5 iunie 2024.
- [6] Natalie Schooner, *Frontend vs Backend*,
<https://blog.boot.dev/backend/frontend-vs-backend-meaning/>, Data accesării: 9 iunie 2024.
- [7] HTML, <https://developer.mozilla.org/en-US/docs/Web/HTML>, Data accesării: 5 iunie 2024.
- [8] CSS, <https://developer.mozilla.org/en-US/docs/Web/CSS>, Data accesării: 5 iunie 2024.
- [9] JavaScript, <https://developer.mozilla.org/en-US/docs/Web/JavaScript> Data accesării: 5 iunie 2024.
- [10] Andrew E. Soltis, *Chess*, <https://www.britannica.com/topic/chess>, Data accesării: 30 mai 2024.
- [11] Aruna Pattam, *Artificial Intelligence defined in simple terms*,
<https://www.hcltech.com/blogs/artificial-intelligence-defined-simple-terms>, Data accesării: 12 iunie 2024.
- [12] Chess.com, <https://www.chess.com/terms/elo-rating-chess>, Data accesării: 5 iunie 2024.
- [13] Angelo Gentile III, *Basics of Django: Model-View-Template (MVT) Architecture*,
<https://angelogentileiii.medium.com/basics-of-django-model-view-template-mvt-architecture-8585aecffbf6>, Data accesării: 12 iunie 2024.
- [14] W3Schools, *Introduction to Django*,
https://www.w3schools.com/django/django_intro.php, Data accesării: 9 iunie 2024.
- [15] Paul Kirvan, Ivy Wigmore, Controller,
<https://www.techtarget.com/whatis/definition/controller>, Data accesării: 12 iunie 2024
- [16] SQLite, <https://sqlite.org/> Data accesării: 12 iunie 2024.
- [17] Django-UserCreationForm, <https://www.javatpoint.com/django-usercreationform>, Data

accesării: 8 iunie 2024.

[18] *HTTP Requests*, <https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>, Data accesării: 12 iunie 2024.

[19] *django.contrib.auth*, <https://docs.djangoproject.com/en/5.0/ref/contrib/auth/>, Data accesării: 8 iunie 2024.

[20] *ModelForm*, <https://docs.djangoproject.com/en/5.0/topics/forms/modelforms/>, Data accesării: 9 iunie 2024.

[21] *Imagini table de șah*, https://en.wikipedia.org/wiki/Rules_of_chess, Data accesării: 12 iunie 2024.

[22] *The Django template language: for Python programmers*, <https://docs.djangoproject.com/en/5.0/ref/templates/api/>, Data accesării: 1 iunie 2024.

[23] *Imagini piese de șah populare pe Lichess*,
<https://github.com/lichess-org/lila/tree/master/public/piece/cburnett>, Data accesării: 31 mai 2024.

[24] *EventTarget: addEventListener() method*,
<https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener>, Data accesării: 12 iunie 2024.

[25] *Document: querySelectorAll() method*,
<https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelectorAll> Data accesării: 12 iunie 2024.

[26] *Django Channels*, <https://channels.readthedocs.io/en/latest/>, Data accesării: 2 iunie 2024.

[27] *The WebSocket API (WebSockets)*,
https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API, Data accesării: 2 iunie 2024.

[28] *Difference Between ASGI and WSGI in Django*,
<https://www.geeksforgeeks.org/difference-between-asgi-and-wsgi-in-django/>, Data accesării: 2 iunie 2024.

[29] *Window: sessionStorage property*,
<https://developer.mozilla.org/en-US/docs/Web/API/Window/sessionStorage>, Data accesării: 2 iunie 2024.

[30] *Official Stockfish Github Repo*, <https://github.com/official-stockfish/Stockfish>, Data accesării: 10 iunie 2024.

[31] *Official Stockfish Download Page*, <https://stockfishchess.org/download/>, Data accesării: 10 iunie 2024.

[32] *python-chess: a chess library for Python*, <https://python-chess.readthedocs.io/en/latest/> Data accesării: 10 iunie 2024.

[33] *JSON*, <https://www.json.org/json-en.html>, Data accesării: 12 iunie 2024.