ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Alba Regia Műszaki Kar

Geoinformatikai/Mérnöki/
Természettudományi és Szoftvertechnológiai Intézet

PROJEKTMUNKA 2

Módosított sakkjáték fejlesztése Laravel és React környezetben

OE-AMK

Hallgatók nevei: Inzsöl Mátyás, Gasparics Dániel

Tartalomjegyzék

1. BEVEZETÉS

A sakk az emberi kultúra egyik legősibb és legnemesebb stratégiai játéka, amely több mint másfél évezrede áll a logikai gondolkodás és a taktikai tervezés középpontjában. Gyökerei az indiai chaturanga játékig nyúlnak vissza, amely a 6. században alakult ki, majd az arab shatranj közvetítésével jutott el Európába, ahol fokozatosan formálódott a ma ismert modern sakká. A játék célja nem csupán az ellenfél királyának legyőzése, hanem a stratégiai előrelátás, a döntéshozatal és az analitikus gondolkodás fejlesztése is. Nem véletlenül nevezik sokan a sakkot „a királyok játékának", hiszen egyszerre művészet, tudomány és intellektuális kihívás.

A digitális korszak megjelenésével a sakk túllépett a hagyományos táblás formán, és új dimenziókban éledt újjá. Online platformokon, mobilalkalmazásokban és mesterséges intelligencia alapú rendszerekben vált elérhetővé, lehetővé téve, hogy a játékosok a világ bármely pontjáról összemérjék tudásukat. A modern informatikai technológiák révén a sakk ma már nem csupán szórakozás, hanem kutatási és fejlesztési terep is: az algoritmusok, az adatfeldolgozás és a hálózati kommunikáció szempontjából egyaránt kiváló modell.

A klasszikus szabályrendszer ugyan precízen kidolgozott és évszázadok óta változatlan, ám éppen ez a kötöttség gátat szabhat az új játékmenetek és ötletek kipróbálásának. E felismerés hívta életre a jelen projektet, amelynek célja egy módosított sakkjáték megalkotása, ahol a sakk logikai alapjai megmaradnak, ugyanakkor a játékszabályok, a táblaméret és a figurák konfigurálhatók.

Összességében a projekt célja annak bemutatása, hogyan lehet a klasszikus sakkot korszerű technológiai környezetben újragondolni. A fejlesztett alkalmazás példát ad arra, miként ötvözhetők a hagyományos stratégiai alapelvek a modern webes technológiákkal, és hogyan valósítható meg egy olyan valós idejű, rugalmas szabályrendszerű sakkplatform, amely egyszerre oktatási eszköz, kísérleti rendszer és szórakoztató alkalmazás.

2. KÖVETELMÉNY SPECIFIKÁCIÓ (FUNKCIONÁLIS ÉS NEM FUNKCIONÁLIS)

2.1 Funkcionális követelmények

A funkcionális követelmények a rendszer azon tulajdonságait határozzák meg, amelyek a játék tényleges működéséhez, a felhasználói interakcióhoz és a hálózati kommunikációhoz szükségesek.

2.1.1 Játéktípus és szabályrendszer beállítása

A fejlesztett rendszer egyik legfontosabb funkcionális eleme a játéktípus és szabályrendszer konfigurálhatósága. A cél az volt, hogy a felhasználók ne csupán a klasszikus sakk szabályai szerint játszhassanak, hanem különböző, egyedi variációkat is kipróbálhassanak. Ennek érdekében a játék beállítási modulja teljes mértékben dinamikus és bővíthető módon készült el.

A felhasználó a játék indítása előtt egy grafikus kezelőfelületen választhatja ki a kívánt jellemzőket. Ezek közé tartozik a játék típusa és a tábla mérete.

A rendszer több előre definiált játékmódot tartalmaz, amelyek közül a felhasználó tetszőlegesen választhat a játék indításakor, ezek a következők:

- Alap sakk: a hagyományos 8x8-as tábla és szabályrendszer alapján működik, minden bábu a megszokott helyén és mozgásmintával szerepel.
- Klasszikus paraszthábor: Aki a gyalogjával eléri a másik oldalt, az a győztes. Akkor is nyerhet a játékos, ha leüti ellenfele összes gyalogját, vagy számára már nem marad érvényes lépési lehetőség.
- Egyedi módok:
    - Vezér a 8 gyalog ellen
    - Bástya 5 gyalog ellen
    - Futó három gyalog ellen
    - 2 huszár 3 gyalog ellen

- Vezér a huszár ellen
- Királyvadászat

Fejlesztés alatt álló változatok:

- Aktív sakk (9x8)
- Távoli sakk (8x9)
- Mikrosakk (4x5)

A szabályrendszer a játék logikai motorjában (ChessEngine.js) kerül meghatározásra. Ez a modul kezeli a bábuk mozgását, az ütések érvényességét, a promóciókat, valamint a speciális szabályokat (például a sáncolást vagy az „en passant” lépést). A módosított játéktípusok esetében rugalmasan paraméterezhető, így egyes szabályok kikapcsolhatók vagy módosíthatók — például engedélyezhető a gyalogok „ugrása”, vagy megváltoztatható a bábuk induló pozíciója.

A konfiguráció során megadott adatok a szerveren keresztül jutnak el a másik játékoshoz, ahol automatikusan betöltődnek a felhasználói felületre. Ezzel a rendszer biztosítja, hogy a játék indítása előtt mindkét fél ugyanazt a szabályrendszert és felállást használja.

A játéktípus beállítása nem csupán a játékmenetet határozza meg, hanem a nyerési feltételek logikáját is. Míg a klasszikus sakkban a győzelem a király mattolásával történik, addig egy módosított játékmódban például a következő feltételek is meghatározhatók:

- az összes ellenséges bábu leütése
- egy adott bábu túlsó oldalára juttatása
- elérni, hogy a másik játékos ne tudjon szabályos lépést tenni

A rendszer a győzelmi feltételeket minden érvényes lépés után automatikusan ellenőrzi.

A konfigurálhatóság lehetősége nemcsak játékosok, hanem fejlesztői szempontból is előnyös. A kódszerkezet összetett felépítése miatt a jövőben új játékmódok vagy speciális szabályok is egyszerűen hozzáadhatók a rendszerhez anélkül, hogy a meglévő logika átírásra szorulna.

2.1.2 Multiplayer kapcsolat és kommunikáció

A sakkjáték egyik legfontosabb funkciója a valós idejű, két játékos közötti online kapcsolat megvalósítása.

A React projekt két fő egységből áll:

- a server mappában található Node.js szerver valósítja meg a hálózati logikát,
- míg a client mappa felel a felhasználói felületért és a játékosok interakciójáért.

A kommunikáció Socket.IO technológián keresztül történik, amely egy eseményalapú rendszer. Ennek előnye, hogy a kapcsolat kétirányú és valós idejű, vagyis a szerver és a kliens egyszerre képes üzeneteket küldeni és fogadni, polling (folyamatos lekérdezés) nélkül. Ez különösen fontos egy multiplayer játék esetében, ahol minden ezredmásodperc számít a játékélmény szempontjából.

Kapcsolódási folyamat

A kapcsolat létrehozása a kliens oldaláról indul:

A játékos a böngészőben futó React alkalmazásból csatlakozik a szerverhez, amely a Socket.IO klienskönyvtár segítségével történik.

A szerver minden új kapcsolódáskor egyedi azonosítót rendel a játékoshoz (socket.id), majd elhelyezi őt egy játékszobába.

A játékszoba egy olyan logikai egység, amelyhez pontosan két játékos tartozhat: a fehér és a fekete.

A szobák kezelését a szerver memóriában végzi, így nincs szükség adatbázisra. Új játék létrehozásakor a szerver egy véletlenszerű UUID (Universal Unique Identifier) azonosítót generál, amely a játék szobáját jelöli. A játékosok azonosítói, a tábla állapota, az aktuális körszín és a játéktípus mind a szerveroldali objektumban kerülnek tárolásra.

A szerver tipikus eseményei a következők:

- createRoom – új játékszoba létrehozása és az azonosító visszaküldése a kliensnek,
- joinRoom – második játékos csatlakoztatása a meglévő szobához,
- move – lépésadatok fogadása és továbbítása az ellenfél kliensének,

- drawOrMate – döntetlen, matt vagy játék vége esemény közvetítése,
- disconnect – játékos kapcsolatának megszakadását kezelő esemény.

Kommunikációs elvek és eseménykezelés

A rendszer teljes kommunikációja eseményalapú. Ez azt jelenti, hogy a szerver és a kliens nem folyamatosan kérdezgeti egymást, hanem csak akkor továbbít üzenetet, ha valamilyen esemény történik (például lépés, új játékos csatlakozása vagy játék vége).

Például egy lépés kommunikációja így zajlik:

- A játékos a táblán kijelöli a bábut és a célmezőt.
- A kliens a lépést elküldi a szervernek egy move esemény formájában.
- A szerver frissíti a szoba adatait (aktuális kör, táblaállapot).
- A szerver azonnal visszaküldi az új állapotot a másik játékos kliensének.
- Mindkét játékos felületén megjelenik az új pozíció.

A rendszer képes kezelni az olyan helyzeteket is, amikor a kapcsolat megszakad. A szerver érzékeli a disconnect eseményt, és ha a másik játékos még aktív, értesíti őt a megszakadásról.

Biztonság és stabilitás

Bár a rendszer nem használ adatbázist, a kommunikáció biztonságát több tényező garantálja:

- minden kapcsolat egyedi azonosítón keresztül történik,
- a szobák zártak, így harmadik fél nem tud csatlakozni,
- a szerver csak ismert eseményeket fogad el, más üzeneteket elutasít.

2.1.3 Játékmenet vezérlése és szinkronizáció

A sakkjáték valós idejű működésének kulcsa a játékmenet helyes vezérlése és a két kliens közötti állapotok folyamatos szinkronizálása. A rendszer célja, hogy a játék minden pillanatában csak az egyik játékos legyen lépésre jogosult, a lépések érvényességét a logikai motor automatikusan ellenőrizze, és az eredményt mindkét oldalon azonnal megjelenítse.

Játékmenet vezérlésének alapelvei

A játékmenetet a kliensoldali logikai modul (a ChessEngine.js fájl) irányítja. Ez az egység tartalmazza az összes szabályt, a bábutípusok mozgásmintáit, valamint az érvényes lépések kiszámítását. Amikor a játékos egy bábut kijelöl, a motor ellenőrzi, hogy a kiválasztott bábu a szabályoknak megfelelően léphet-e az adott mezőre. Érvénytelen lépés esetén a rendszer nem engedi végrehajtani a műveletet.

A szerver a lépések logikáját nem értelmezi, csak az adatátvitelt biztosítja. Ha a kliens érvényes lépést hajt végre, az esemény (move) elküldésre kerül a szerverhez, amely azt a másik játékoshoz továbbítja.

A vezérlés szempontjából a rendszer minden körben a következő lépéseket hajtja végre:

1. Aktív játékos meghatározása – a rendszer nyilvántartja, hogy melyik szín (fehér vagy fekete) következik.
2. Lépés kiválasztása és ellenőrzése – a getValidMoves() függvény kiszámítja a bábu számára érvényes célmezőket.
3. Lépés végrehajtása – ha a lépés szabályos, a kliens elküldi az adatokat a szervernek.
4. Állapotfrissítés – a szerver a módosított táblát és körinformációt azonnal továbbítja a másik kliensnek.
5. Következő kör indítása – a körszín váltása után az ellenfél lesz aktív.

Ezzel a folyamat minden lépés után megismétlődik, biztosítva a folyamatos, felváltva történő játékmenetet.

Szinkronizáció és adatkezelés

A táblaszinkronizáció központi szerepet tölt be a rendszerben. Minden lépés után a szerver memóriában tárolja a játék pillanatnyi állapotát – a tábla pozícióit, a körszínt, a játéktípust és a játék állapotát (döntetlen, matt stb.).

Minden lépés, új játék indítása, döntetlen felismerése vagy játszma lezárása külön eseményként kerül továbbításra.

A főbb események:

- move – lépés végrehajtása és frissítése,
- updateBoard – új táblaállapot továbbítása a másik kliens felé,
- drawOrMate – a játék végének közvetítése,

2.1.4 Nyerés, vesztés és patthelyzet felismerése

A sakkjáték egyik legfontosabb része a játszma lezárásának kezelése, azaz annak felismerése, hogy mikor következik be győzelem vagy döntetlen. A rendszer feladata, hogy ezek az állapotok automatikusan felismerhetők legyenek.

Általános működési elv

A játékmenet minden lépése után a rendszer automatikusan meghívja a getGameStatus() függvényt, amely kiértékeli a tábla aktuális állapotát és meghatározza, hogy a játszma folytatható-e vagy lezárult.

Ez a függvény a következő eseteket különbözteti meg:

1. Matt: a király sakkban áll, és nincs szabályos lépés, amellyel elkerülhető lenne a támadás.
2. Patt: a soron következő játékos nem tud szabályos lépést tenni, de a király nincs sakkban – az eredmény döntetlen.
3. Háromszori ismétlődés: ugyanaz az állás háromszor előfordul, ami automatikus döntetlent eredményez.
4. Speciális játéktípusok esetén: egyedi győzelmi feltételek teljesülése (például adott bábu eléri a tábla túlsó oldalát, vagy az ellenfél összes bábujának elvesztése).

Különböző játéktípusokhoz tartozó feltételek

A rendszer nemcsak a klasszikus sakkban ismert szabályokat, hanem a módosított játéktípusokhoz tartozó egyedi végállapotokat is kezeli.

Kommunikáció a játék lezárásakor

Amikor a rendszer felismeri, hogy a játék véget ért, a kliens egy eseményt küld a szervernek, amely azonnal továbbítja azt a másik játékos felé. Az esemény tartalmazza a játék kimenetelét (például *white wins*, *black wins* vagy *draw*), valamint egy rövid üzenetet, amely a felhasználói felületen jelenik meg. A kliens ezután letiltja a további lépések lehetőségét, és megjeleníti a végeredményt, például:

„ Black wins – Black cannot move.”

„Draw by fifty-move rule”

„White wins – all pawns have been taken!”

Ezzel a megoldással a program megbízhatóan képes kezelni a sakk és annak különféle variánsainak végállapotait, legyen szó klasszikus partikról vagy teljesen új, kísérleti játékmódokról.

2.1.5 Játéktörténet és felhasználói interakció

A játéktörténet és a felhasználói interakció kezelése kulcsfontosságú szerepet játszik a sakkjáték használhatóságában és átláthatóságában.

Játéktörténet kezelése

A rendszer minden lépést automatikusan rögzít a memóriában tárolt *history* tömbben, amely a React komponensen belül (Game.js) él, és nem kerül adatbázisba mentésre.
A játéktörténet minden elem egy objektum, amely a következő adatokat tartalmazza:

- a lépést végrehajtó játékos színe (wTurn / turnColor),
- az induló és célmező azonosítója (pl. "e2" → "e4"),
- az elmozdított bábu típusa (piece),
- ha történt ütés, az elütött bábu adatai is bekerülnek a captured mezőbe,
- a lépés sorszáma (az index alapján generált),
- valamint a lépés időpontja (timestamp).

A kliens minden lépés után frissíti a *history* tömböt egy új bejegyzéssel.

Felhasználói interakció és felület

A felhasználói interakció teljes mértékben a React frontend komponensein keresztül valósul meg. A játékos az egér segítségével jelöli ki a mozgatni kívánt bábut és a célmezőt.
A felület vizuálisan visszajelzést ad:

- a kijelölt bábu mezője megvilágítódik,
- érvénytelen lépés esetén a mező nem reagál, és hibaüzenet nem jelenik meg, csak a kijelölés törlődik.

A rendszer minden egyes lépésnél ellenőrzi, hogy melyik játékos van soron, és csak az aktív fél számára engedélyezi a lépéseket. Ha a játék véget ér (matt, patt vagy döntetlen), a felület automatikusan zárolja a mezőket, megakadályozva a további interakciót.

A játékmenethez tartozó üzenetek – például „Current turn”, „Game type: Active Chess”, – a felhasználói felület tetején, jól látható helyen jelennek meg. A rendszer képes valós időben reagálni a szerver által küldött eseményekre, így a játékosok azonnal látják a végeredményt.

Interaktív elemek

A felhasználói felület tervezése során cél volt a minimalista, de informatív megjelenítés. A tábla felett elhelyezett játékos-információk (név, szín) egyaránt dinamikusan frissülnek.

A játéktörténet és felhasználói interakció modulja biztosítja a játék átláthatóságát és valós idejű visszajelzését. A megvalósítás nemcsak kényelmes és intuitív felhasználói élményt nyújt, hanem a fejlesztés és hibakeresés szempontjából is nélkülözhetetlen funkciókat biztosít.

2.2 Nemfunkcionális követelmények

2.2.1 Teljesítmény és válaszidő

A sakk valós idejű, interaktív jellegéből adódóan a teljesítmény kiemelten fontos tényező. A játékosok közötti kommunikációnak késleltetés nélkül kell történnie, hogy a játékmenet folyamatos és természetes maradjon.

A fejlesztés során cél volt, hogy az adatok feldolgozása és a lépések továbbítása valós időben, észrevehető késés nélkül történjen, így a felhasználók élménye megfeleljen a hagyományos offline sakk dinamikájának.

2.2.2 Biztonság és hálózati stabilitás

A rendszer működéséhez folyamatos hálózati kapcsolat szükséges, mivel a játékmenet és az állapotváltozások valós időben kerülnek továbbításra. A kapcsolat megbízhatóságát a Socket.IO beépített újracsatlakozási mechanizmusa biztosítja: ha egy játékos internetkapcsolata megszakad, a kliens automatikusan megkísérli a szerverhez való újbóli kapcsolódást.

A kommunikáció biztonságát több tényező is garantálja:

- minden kliens egyedi azonosítót kap (socket.id), amely alapján a szerver elkülöníti a kapcsolatokat,
- a játékszobákhoz csak meghívóval (room ID) lehet csatlakozni, így illetéktelen felhasználó nem léphet be,
- a szerver kizárólag előre definiált eseményeket (pl. move, drawOrMate) fogad, így idegen üzenetek nem dolgozhatók fel.

Mivel a rendszer nem használ adatbázist, a hálózati stabilitás és a memóriában tárolt adatok kezelése különösen fontos. A szerver minden új kapcsolat létrejöttekor frissíti a játékállapotokat, és megszakítás esetén is képes felismerni, ha egy játékos elhagyta a szobát.

2.2.3 Karbantarthatóság és bővíthetőség

A fejlesztés során kiemelt szempont volt, hogy a rendszer könnyen fejleszthető és skálázható maradjon. A Laravel + React architektúra tisztán elkülöníti az alkalmazás logikai és megjelenítési rétegeit:

- a Laravel kezeli a szerveroldali kommunikációt és a Socket.IO integrációt,
- a React biztosítja a kliensoldali logikát és a felhasználói felületet.

A projekt moduláris szerkezetben épül fel: minden funkció külön komponensként valósul meg, így a későbbi fejlesztések (például új játéktípusok, egyéni szabályrendszerek, statisztikai modulok) hozzáadása nem igényli a teljes rendszer újraírását.

A kód kommentált és strukturált, ami megkönnyíti az utólagos módosításokat és a csapatmunkát. A játékmotor külön fájlban (ChessEngine.js) található, így más projektekbe is átültethető, vagy akár önálló könyvtárként továbbfejleszthető.

A karbantarthatóság szempontjából a memóriában történő állapotkezelés és az adatbázis hiánya egyszerűsíti a rendszer működését: nincs szükség adatkonzisztencia-kezelésre, migrációkra vagy komplex háttérműveletekre. A bővíthetőség pedig biztosított azáltal, hogy az események (move, joinRoom, drawOrMate) dinamikusan bővíthetők új logikai ágakkal.

3. TERVEZÉS (VAGY RENDSZERTERV)

3.1 Architektúra áttekintése

A fejlesztett sakkjáték kliens–szerver alapú architektúrát követ, amely a valós idejű kommunikáció megvalósítására optimalizált.

A rendszer három fő rétegből épül fel:

1. Frontend (React):

   A felhasználói felület megvalósításáért felelős réteg, amely megjeleníti a táblát, kezeli a felhasználói interakciókat, és biztosítja a játékosok számára az élő kapcsolatot a szerverrel.

2. Backend (Laravel):

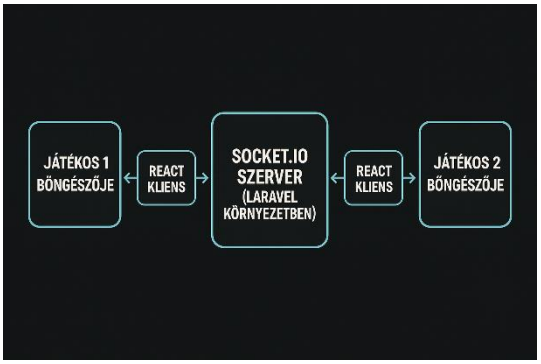
   A szerveroldali logikát és a Socket.IO kommunikációt vezérli. A Laravel itt nem hagyományos adatkezelési célokat lát el, hanem keretrendszerként szolgál a webszerver működéséhez és a React alkalmazás futtatásához a VPS-en.

3. Kommunikációs réteg (Socket.IO):

   A két játékos közötti valós idejű adatátvitelért felelős rendszer. Ez a komponens biztosítja, hogy minden lépés és esemény azonnal megjelenjen a másik játékosnál.

A teljes rendszer egy VPS szerveren fut, ahol a Laravel alkalmazás szolgálja ki a React frontend alkalmazást és a Socket.IO szervert.

A szerkezet leegyszerűsítve így írható le:



Az architektúra előnye, hogy nincs szükség adatbázisra: a játékállapot, lépések és játékosinformációk memóriában kerülnek tárolásra.

3.2 Backend (Laravel) komponensek

A backend réteg a Laravel keretrendszerre épül, amely biztosítja az alkalmazás szerveroldali futtatását és a Socket.IO kiszolgálását. A Laravel alkalmazás fő feladata a React kliens kiszolgálása és a Node.js alapú kommunikációs szerver futtatása.

A legfontosabb komponensek:

- server.js

  A Node.js alapú szerver, amely a Socket.IO események kezeléséért felelős. Ez biztosítja a szobák (rooms) létrehozását, a játékosok csatlakoztatását, a lépések továbbítását és a kapcsolat megszakadásának kezelését.

- App.js

  A Laravel projekt inicializáló komponense, amely a React alkalmazás betöltését végzi és a környezet konfigurálását kezeli.

3.3 Frontend (React) komponensek

A frontend réteg a felhasználói élményt és az interaktivitást biztosítja. A React alkalmazásban minden funkció külön komponensbe van szervezve.

A főbb komponensek:

- Game.js – A játék fő logikáját és megjelenítését kezeli. Felelős a sakkmotor inicializálásáért, a Socket.IO kliens kapcsolat létrehozásáért, a lépések küldéséért és fogadásáért.
- ChessEngine.js – A játékmotor, amely kezeli a bábuk mozgását, az érvényes lépéseket, a szabályok ellenőrzését, valamint a győzelmi feltételek megállapítását.
- CustomBoard.js – A tábla vizuális megjelenítéséért felelős komponens. Interaktív mezőket renderel, kiemeli a kijelölt bábukat és a lehetséges célmezőket.
- InitGame.js – A játék indítását és a beállításokat vezérli. Itt választható ki a játéktípus és a tábla mérete.
- CustomDialog.js – Felugró ablakokat kezel, például játék befejezése vagy döntetlen esetén.
- socket.js – A Socket.IO klienskonfigurációját tartalmazza, amely összekapcsolja a React alkalmazást a szerverrel.

A React komponensek állapotvezéreltek (state-based), ami biztosítja, hogy minden változás azonnal megjelenjen a felhasználói felületen. A lépések, állapotfrissítések és üzenetek automatikusan újrarenderelik a megfelelő komponenseket, így nincs szükség manuális frissítésre vagy újratöltésre.

3.4 Kommunikációs modell (Socket.IO)

A rendszer kommunikációját a Socket.IO technológia valósítja meg.

A legfontosabb események és feladatuk:

Esemény neve	Küldő	Címzett	Funkció
createRoom	kliens	szerver	Új játék létrehozása, szobaazonosító generálása
joinRoom	kliens	szerver	Második játékos csatlakoztatása egy meglévő szobához
move	kliens	szerver → ellenfél	Lépésadatok továbbítása valós időben
drawOrMate	kliens	szerver → ellenfél	A játék végét jelző esemény
disconnect	szerver	kliens	A kapcsolat megszakadásának jelzése

Ha egy játékos kilép vagy megszakad a kapcsolata, a szerver a disconnect esemény segítségével értesíti a másik felet, és felszabadítja a játékszobát. Ez a modell egyszerű, mégis stabil megoldást kínál a valós idejű kétirányú kommunikációhoz.

3.5 Adatáramlási és folyamatábrák

A rendszer adatáramlása jól elkülöníthető lépésekből áll, amelyek egy-egy játékciklust alkotnak:

1. Kapcsolódás:

   A játékos elindítja a React alkalmazást, megadja a tetszőleges felhasználónevét, amely Socket.IO-n keresztül kapcsolódik a szerverhez. A szerver egyedi azonosítót ad a kliensnek.

2. Szoba létrehozása:

   Az első játékos létrehoz egy új szobát a „START A GAME” gombra kattintva, majd megosztja az azonosítót a második játékossal.

3. Csatlakozás:

   A második játékos belép a szobába a kapott azonosítóval, és ezzel a szerver mindkét kapcsolatot regisztrálja.

4. Játékmenet:

   A játékos lépései (move) a szerveren keresztül azonnal továbbításra kerülnek a másik fél felé.

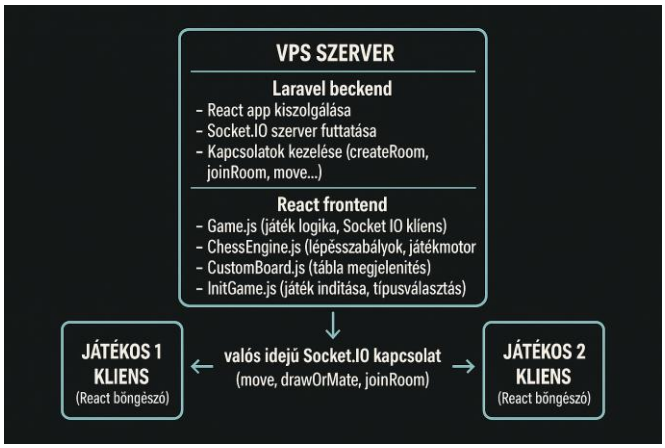
5. Eredmény:

   A kliensoldali motor felismeri a győzelmi vagy döntetlen feltételeket, majd a szerver a drawOrMate esemény segítségével értesíti a feleket.

6. Befejezés:

   A szerver lezárja a szobát, a memóriában tárolt adatokat törli, és új játék indítható.

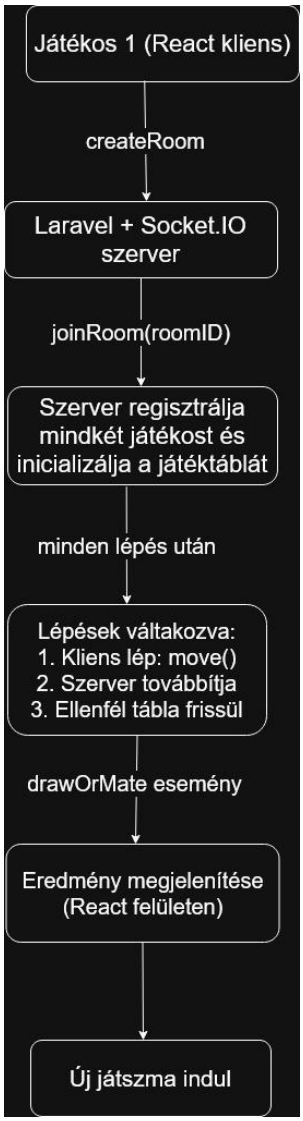
Architektúra diagram



Ez az architektúra központi szerveres topológiát követ.

Folyamatábra – Játékmenet és adatáramlás

Az alábbi folyamatábra a játék teljes életciklusát szemlélteti az első kapcsolatfelvételtől a játszma lezárásáig:



4. MEGVALÓSÍTÁS

A megvalósítás során a cél egy olyan webalapú sakkalkalmazás létrehozása volt, amely képes valós idejű kommunikációra két játékos között, és lehetővé teszi a játékszabályok, táblaméretek és bábutípusok dinamikus konfigurálását. A rendszer felépítése a modern webfejlesztés bevált technológiáira épül, a fejlesztés során külön figyelmet kapott az egyszerűség, a skálázhatóság és a karbantarthatóság.

4.1 Fejlesztői környezet és technológiák

A fejlesztés során egy VPS (Virtual Private Server) környezetben telepített rendszer került kialakításra, amely Linux-alapú (Ubuntu) operációs rendszeren fut. A szerveroldali környezetben a Laravel és a Node.js keretrendszerek együttesen biztosítják az alkalmazás működését, míg a

kliensoldalon a React könyvtár gondoskodik a dinamikus megjelenítésről és az interaktivitásról.

Alkalmazott technológiák

- Laravel 10 (PHP 8.x) – a szerveroldali keretrendszer, amely a webes infrastruktúrát biztosítja és a Socket.IO integrációt futtatja.
- Node.js + Socket.IO – valós idejű kommunikáció megvalósításához használt környezet.
- React 18 – a frontenden használt JavaScript-könyvtár a komponensalapú felépítéshez.
- Vite – gyors fejlesztői szerverként és buildrendszerként szolgál a React alkalmazás számára.
- Visual Studio Code – a fejlesztés fő integrált környezete (IDE).
- Nginx / Apache – a Laravel alkalmazás kiszolgálásához használt webkiszolgáló.
- Git / GitHub – verziókezelésre és forráskód-tárolásra.

4.2 Szerver–kliens kapcsolat létrehozása

A rendszer kommunikációját a Socket.IO technológia biztosítja, amely a WebSocket protokollra épülve teszi lehetővé a valós idejű, kétirányú adatcserét a kliens és a szerver között. A kapcsolat inicializálása a kliensoldali React alkalmazásban történik, ahol a *socket.js* fájl felel a szerverrel való összeköttetés létrehozásáért. A szerver oldali kommunikációt a *server.js* fájl kezeli, amely Node.js környezetben működik, párhuzamosan a Laravel alapú webalkalmazással. Amikor egy felhasználó csatlakozik, a szerver automatikusan egyedi azonosítót (*socket.id*) rendel hozzá, majd szükség esetén létrehozza a játékhoz tartozó szobát. A játékosok ezután a *createRoom* és *joinRoom* eseményeken keresztül kapcsolódhatnak ugyanahhoz a játékmenethez.

A legfontosabb üzenettípusok:

- move – a játékos lépését továbbítja az ellenfélnek,
- drawOrMate – a játék végét jelző üzenet,
- disconnect – a kapcsolat megszakadásának kezelése.

A kapcsolat kétirányú, tehát a kliens nemcsak üzenetet küld, hanem folyamatosan figyeli is a szervert az állapotváltozásokra.

4.3 Játékmenet logika és eseménykezelés

A játékmenet logikai feldolgozását a kliensoldali motor végzi, amelyet a ChessEngine.js fájl tartalmaz. Ebben a modulban találhatók a szabályrendszer meghatározásai.

A játék logikai folyamata minden körben az alábbi lépéseket követi:

1. Lépés kiválasztása:

   A játékos kijelöli a mozgatni kívánt bábut és a célmezőt. A rendszer azonnal ellenőrzi, hogy a lépés érvényes-e a játéktípus szabályai szerint.

2. Lépés validálása:

   Ha a lépés érvényes, a makeMove() függvény módosítja a tábla állapotát, és meghívja a megfelelő függvényt a játék végállapotának ellenőrzésére.

3. Állapotfrissítés:

   A frissített állapot elküldésre kerül a szervernek socket.emit("move", data) formájában, amely azonnal továbbítja azt az ellenfél kliensének.

4. Játék vége:

   Amennyiben a getGameStatus() függvény mattot, pattot vagy speciális győzelmet észlel, a kliens eseményt küld a szervernek, amely értesíti a másik játékost a végeredményről.

Az eseménykezelés teljesen aszinkron módon történik, így a játékmenet folyamatos és akadásmentes marad.

4.4 Felhasználói felület megvalósítása

A felhasználói felületet a React biztosítja, amely komponensalapú megközelítést alkalmaz. A cél egy egyszerű, letisztult és reszponzív felület kialakítása volt, ahol a játékosok intuitív módon tudnak lépni és követni a játékmenetet.

A fő vizuális elemek:

- Sakk tábla – a CustomBoard.js komponens rendereli a 8×8-as (vagy módosított méretű) táblát, amely a mezőket és a bábukat Unicode karakterekkel jeleníti meg.
- Játékinformációs sáv – megjeleníti, melyik játékos következik

- Modal popup ablakok – döntetlen vagy játék vége esetén felugró üzenetet jelenít meg.
- Játéktípus – megjeleníti, hogy milyen játéktípusban játszanak a játékosok
- Room ID – ezzel az egyedi azonosítóval tud a második játékos csatlakozni a játékszobához
- Játékosok (Players) – az első játékos felhasználóneve és a második játékos csatlakozása után megjelenik mindegyikük felhasználóneve

A React useState és useEffect hookjai biztosítják a komponensek közötti állapotkezelést és a valós idejű frissítést. A Socket.IO kliens közvetlenül integrálva van a React komponensekbe, így a kommunikációs események automatikusan újrarenderelik a felületet.

A stílus CSS modulokkal és Material UI elemekkel valósult meg, ami modern, átlátható és reszponzív dizájnt eredményezett.

5. TESZT

A fejlesztett rendszer működésének validálása érdekében többféle tesztelési módszer került alkalmazásra. A cél az volt, hogy a szoftver minden komponense — a játékmenet logikája, a kommunikáció, a teljesítmény és a felhasználói felület — megbízhatóan, hiba nélkül működjön különböző körülmények között is.

A tesztelés kiterjedt mind a funkcionális elemekre, mind a rendszerszintű teljesítményre és a multiplayer kommunikáció stabilitására. A fejlesztés iteratív módon zajlott: minden új funkció beépítése után manuális tesztek futottak le, hogy a hibák azonnal javíthatók legyenek.

5.1 Funkcionális tesztek

A funkcionális tesztek célja annak ellenőrzése volt, hogy a rendszer megfelel-e a specifikált követelményeknek, és a játékmenet a tervezett szabályok szerint működik-e.

Tesztelt funkciók:

- Játéktípus beállítása: a játékos képes volt különböző játékmódokat (alap sakk, parasztháború stb.) kiválasztani a kezdőképernyőn.
- Lépések érvényesítése: a ChessEngine.js megfelelően felismerte az érvényes és érvénytelen lépéseket; érvénytelen lépés esetén nem történt táblaállapot-változás.

- Játékállapot frissítése: minden sikeres lépés után a tábla azonnal frissült, és a soron következő játékos színe helyesen váltakozott.
- Győzelem, matt, patt felismerése: a fontos metódusok, függvények helyesen azonosították a végállapotokat, és a felhasználói felületen megjelent a megfelelő üzenet.

Tesztelési módszer:

A funkcionális tesztelés főként manuálisan történt, két böngészőlap futtatásával, amelyek két különböző játékost szimuláltak. Ezen kívül a Firefox fejlesztői konzoljában figyelve lettek a hálózati események, hogy az adatátvitel minden lépésnél megtörtént-e.

A tesztek eredménye azt mutatta, hogy voltak olyan funkciók a fejlesztéskor amelyeket jobban át kellett szabni.

5.2 Teljesítménytesztek

Tesztelési környezet:

- VPS szerver (6 vCPU, 11,8 GB RAM)
- Kliens böngésző: Firefox
- Hálózati kapcsolat: 918,79 Mbit/s letöltés, 774,40 Mbit/s feltöltés

6. ÖSSZEFOGLALÁS

A projekt célja egy valós idejű, többjátékos módosított sakkjáték megvalósítása volt, amely lehetővé teszi a szabályrendszer, a játéktábla mérete és a használt bábuk szabad konfigurálását. A rendszer a hagyományos sakk alapelveit ötvözi a modern webes technológiákkal, így nemcsak szórakoztató és interaktív alkalmazás született, hanem egy fejlesztési és oktatási szempontból is hasznos szoftver, amely a hálózati kommunikáció, az eseményvezérelt architektúra és a kliens–szerver logika gyakorlati megvalósítását demonstrálja.

Jövőbeli fejlesztési lehetőségek

A jelenlegi rendszer jól működő prototípus, amely számos irányban továbbfejleszthető:

- Adatbázis integráció: játékstatisztikák, felhasználói profilok és ranglisták tárolásához.
- AI (mesterséges intelligencia) ellenfél: gépi algoritmus implementálása egyjátékos módhoz.

- Játékmód-bővítés: új szabályrendszerek és egyedi sakkvariánsok, például háromjátékos vagy 10×10-es táblák bevezetése.
- Grafikai fejlesztések: animációk, 3D-s nézet vagy témaválasztó megjelenése.
- Mobiloptimalizálás: a React frontend továbbfejlesztése PWA (Progressive Web App) irányba, hogy a játék mobilról is futtatható legyen.
- Érvényes célmezők megjelenítése egy bábura való kattintáskor
- A figurák lépéseit kísérő hangjelzés lejátszása
- Időkorlátos játszmák (pl. 5 perces blitz mód)
- Statisztikák és grafikus elemzések a játéktörténet alapján
- Kilépés gomb beépítése a játék egyszerű bezárásához

A projekt során egy olyan webes alkalmazás készült, amely bemutatja, miként ötvözhetők a klasszikus sakk elvei a modern webfejlesztés technológiai megoldásaival. A végeredmény egy valós idejű, interaktív és szabályrendszerében szabadon konfigurálható sakkjáték, amely képes a játékosok számára új élményt nyújtani, miközben a fejlesztési szempontból is jól strukturált és karbantartható rendszert testesít meg. A dolgozat célkitűzései teljesültek, a rendszer működőképes, és alapot biztosít a további fejlesztésekhez — mind technológiai, mind kutatási szinten.

7. IRODALOMJEGYZÉK

[1] MisterCutie, “History of Chess: The Basics”, [Online] Available: https://www.chess.com/article/view/the-history-of-chess [Hozzáférés dátuma: 29. október 2025].

[2] "INTRODUCING THE NEW CHALLENGES IN THE PLAY MENU", [Online] Available: https://learningchess.net/blog/?p=2661#play_chess_huM5 [Hozzáférés dátuma: 25 október 2025]