**Eötvös Loránd Tudományegyetem**

**Informatikai Kar**

**Informatikatudományi Intézet**

**Média- és Oktatásinfromatika Tanszék**

Online absztrakt stratégiai társasjáték megvalósítása

Szerző: Témavezető:

Gyepes Gergő Korom Szilárd

Programtervező informatikus BSc. Doktorandusz

**Budapest, 2023**

Ide kerül a hivatalos témabejelentő lap.

Tartalomjegyzék

[**1.** **Bevezetés** 1](#_Toc167915780)

[**2.** **Felhasználói dokumentáció** 2](#_Toc167915781)

[**2.1.** **Megoldott probléma rövid leírása** 2](#_Toc167915782)

[**2.2.** **Az oldal elérése** 2](#_Toc167915783)

[**2.3.** **Az oldal felépítése.** 2](#_Toc167915784)

[**2.3.1.** **Navigációs sáv** 2](#_Toc167915785)

[**2.3.2.** **Bejelentkezési/regisztrációs felület.** 3](#_Toc167915786)

[**2.3.3.** **Fő oldal** 5](#_Toc167915787)

[**2.3.4.** **Szoba** 8](#_Toc167915788)

[**3.** **Fejlesztői dokumentáció** 14](#_Toc167915789)

[**3.1.** **Technológiai Stack** 14](#_Toc167915790)

[**3.2.** **Rendszer** 14](#_Toc167915791)

[**3.3.** **Kezdeti lépések** 14](#_Toc167915792)

[**3.4.** **Környezet felállítása a további fejlesztéshez** 15](#_Toc167915793)

[**3.5.** **Adatbázis (MongoDB)** 16](#_Toc167915794)

[**3.6.** **Socket.IO Integráció** 18](#_Toc167915795)

[**3.7.** **Backend (Express.js)** 19](#_Toc167915796)

[**3.7.1.** **Felépítés** 19](#_Toc167915797)

[**3.7.2.** **Szerver-kliens kommunikáció** 20](#_Toc167915798)

[**3.7.3.** **Kontrollerek** 21](#_Toc167915799)

[**3.8.** **Frontend (React)** 24](#_Toc167915800)

[**4.** **Összefoglalás és további fejlesztési lehetőségek** 43](#_Toc167915801)

[**4.1.** **Összefoglalás** 43](#_Toc167915802)

[**4.2.** **További fejlesztési lehetőségek** 43](#_Toc167915803)

[**5.** **Irodalomjegyzék, hivatkozások** 44](#_Toc167915804)

# **Bevezetés**

A társasjátékok mindig is különleges helyet foglaltak el az életemben. Gyerekként mindig is szívesen töltöttem a szabadidőm barátaimmal és családommal különböző társasjátékok mellett, és felnőttként is szenvedéllyel hódolok ezen hobbinak. A játékok nem csak szórakoztatóak, hanem kiváló lehetőséget nyújtanak a stratégia gondolkodás, a problémamegoldás és csapatmunka fejlesztésére is. A dolgozatom témájának választott Azul az egyik kedvenc játékom és véleményem szerint kellő kihívást nyújt a játék megvalósítása.

Ez a projekt ötvözi a társasjátékok és webfejlesztés iránti érdeklődésemet. Célom, hogy létrehozzak egy olyan online platformot, amely lehetővé teszi a felhasználók számára, hogy valós időben játszhassanak egymással, bárhol is legyenek a világban. A dolgozat nemcsak a szakmai képességeimet fejleszti, teszi próbára, hanem egyúttal egy szórakoztató eszközt biztosít a társasjátékok kedvelői számára.

A projekt célja egy webalkalmazás fejlesztése, amely lehetővé teszi a felhasználók számára, hogy különböző társasjátékokat játszhassanak online, valós időben. A rendszernek biztosítania kell a következő funkciókat:

* **Felhasználói Regisztráció és Bejelentkezés**: A felhasználók regisztrálhatnak egy fiókot, és bejelentkezhetnek az alkalmazásba, hogy hozzáférjenek a játékokhoz.
* **Játékterek Létrehozása és Csatlakozás**: A felhasználók létrehozhatnak új játéktereket (szobákat), vagy csatlakozhatnak meglévő szobákhoz, ahol más felhasználókkal együtt játszhatnak.
* **Valós Idejű Kommunikáció**: A Socket.IO segítségével valós idejű kommunikáció valósul meg a játékosok között, lehetővé téve a zökkenőmentes játékmenetet.
* **Játéklogika Kezelése**: A szerver oldalon fut a játéklogika, amely biztosítja a játékszabályok betartását és a játékmenet folytonosságát.
* **Mentés**: A játék állapotának folyamatos mentése, visszatöltése és a felhasználók adatainak tárolása a MongoDB adatbázisban történik.

# **Felhasználói dokumentáció**

## **Megoldott probléma rövid leírása**

A dolgozatomban egy böngészőben játszható társasjáték megvalósítását tűztem ki célul. A weboldalra a felhasználók be tudnak lépni és vagy regisztrálni. Majd ezután elérik a fő oldalt, ahol egy szoba hosztolás – csatlakozás rendszeren került implementálásra. Egy szobához 1-4 ember tud csatlakozni. A szobákban, az egész életciklusuk alatt, használható egy chat ablak a felhasználók közti kommunikációra. A játékot akkor indíthatjuk, ha legalább ketten vagyunk a szobában és a szoba összes tagja jelezte, hogy készen áll.

Ezután a játék automatikusan elindul. Felváltva vehetünk le csempéket a piacokról és helyezhetjük azokat a táblánkra. A játék az Azul társasjáték szabályait követit. Ha egy felhasználó nincs tisztában a szabályokkal vagy esetleg a weboldal használata nem érthető számára, biztosítottam neki egy súgó-t, amelyet a „?” gombbal meg tud nyitni. Itt lehet elolvasni a szabálykönyvet a játékhoz és az oldal használatához biztosított segédletet.

## **Az oldal elérése**

Sajnos a játék, amiről mintáztam az alkalmazást nem public domain, ezért a weboldal nem lett publikálva. Ha ki szeretnénk próbálni, akkor a 3.4-es lépéseknek megfelelően saját (labor) környezetünkben tehetjük meg azt.

## **Az oldal felépítése.**

Az alkalmazás egy úgy nevezett SPA (single page application), de a könnyen érthetőség kedvéért, az egy böngésző oldalon, egy időben megjelenő felületet oldalnak fogom nevezni. A fejlesztői dokumentációban részletesebben lesz róla szó.

A weboldal elsősorban nagyképernyőre lett optimalizálva, de kisebb kijelzőkkel is kompatibilis. Kis kijelzős üzemmódban bizonyos funkciókat elrejtettünk, ezeket navigációs gombokkal tudjuk majd elérni.

### **Navigációs sáv**

Minden oldalon a lap alján szerepel egy sáv, amelyen megjelenítjük a jelenleg bejelentkezett felhasználó nevét, egy kijelentkezés gombot, egy info/sugó gombot és az oldalunk logóját.



1. ábra navigációs sáv

A kijelentkezés gomb és a felhasználó neve csak a fő és játékszoba oldalakon érhető el.

A súgóra kattintva megnyílik egy panel, ahol elolvashatjuk a játék (Azul) szabálykönyvét és az oldal használati útmutatóját is.

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

2. ábra súgó

### **Bejelentkezési/regisztrációs felület.**

Amikor ellátogatnak a weboldalra, a bejelentkezési felületet fogják először meglátni. Itt, ha már van létrehozva felhasználójuk, tudnak tovább lépni az alkalmazás fő oldalára.

A képen szöveg, képernyőkép, Betűtípus, Téglalap látható

Automatikusan generált leírás

3. ábra bejelentkezési felület

Amennyiben nincsen még felhasználójuk, a bejelentkezés gomb fölötti “Don’t’ have an account?” linkkel tudnak átváltani a regisztrációs felületre.

A képen szöveg, képernyőkép, Betűtípus, tervezés látható

Automatikusan generált leírás

4. ábra regisztrációs felület

A regisztrációhoz szükséges megadni egy egyedi felhasználó nevet, egy legalább 6 karakter hosszú jelszót és a jelszót mégegyszer, ezzel megerősítve azt.

Ha a bejelentkezési adatainkat helyesen kitöltöttük () az alkalmazás a fő oldalra fog minket navigálni.

### **Fő oldal**

A főoldal két fő komponensből áll. Egy űrlapból, ami kitöltésével egy új szobát tudunk nyitni és egy, a jelenlegi szobákat tartalmazó listából.

**Új szoba felvétele**

Az űrlapon elegendő cask a szoba nevét megadni, ekkor bárki csatlakozhat. A Jelszó mező kitöltésével korlátozhatjuk a szobánkhoz való hozzáférést.

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

5. ábra új szoba form

Kisebb kijelző esetén az “Add Password” és “Remove Password” gombokkal tudjuk ki és be kapcsolni a jelszó mező megjelenítését.

A képen szöveg, képernyőkép, Betűtípus, Téglalap látható

Automatikusan generált leírás

6. ábra új szoba from kis kijelzőhöz

**Meglévő szobák**

A job oldali lista tartalmazza a jelenlegi szobákat és egy keresés mezőt, amivel a szobák nevére tudunk szűrni.

A képen szöveg, képernyőkép, tervezés látható

Automatikusan generált leírás

7. ábra szobák listája

Az itt megjelenő kártyák a szobákról tartalmaznak információkat és egy gombot a csatlakozáshoz. A szobáknak megjelenítjük a nevét, jelenlegi állapotát, a csatlakozott felhasználók számát és azt is mutatjuk, hogy szükséges-e jelszó a belépéshez. Amennyiben nem indult még el játék az adott szobában az állapota „Waiting”, ha már megkezdődött a játék, akkor a „Game in progress” feliratot láthatjuk. A szobához csak akkor tudunk csatlakozni, ha még nem indult el a játék és még nem telt be a férőhelyek száma (játékosok száma kevesebb, mint 4). Ha a szobát jelszó védi, azt egy lakat ikonnal jelezzük. Ha erre az ikonra rávisszük a mutatót vagy rákattintunk a csatlakozás gombra, akkor megjelenik a jelszó beviteli mező.

### **Szoba**

Amikor csatlakozunk egy szobához, akár mi hoztuk létre vagy a listából kiválasztva léptünk be, ugyan arra az oldalra navigál minket az alkalmazás. Ezt a szoba oldalt két részre oszthatjuk. A bal oldali panelre és a játéktérre.

Kezdeti állapotban a bal panel tartalmaz két gombot. Az egyikkel el tudjuk hagyni a szobát, a másik gombbal meg tudjuk jelezni játékostársainknak, hogy készen állunk a játék indítására. A gombok alatt található egy chat ablak, amellyel valós időben tudunk kommunikálni a szoba tagjaival.

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

8. ábra baloldali panel

A játék indítása előtt a játéktéren nem a játéktáblát mutatjuk, hanem a szobához csatlakozott felhasználók állapotát.

A képen szöveg, képernyőkép, Téglalap, diagram látható

Automatikusan generált leírás

9. ábra teljes szoba - várakozó

Miután minden játékos „Ready” állapotba került, a játék elindul. A bal oldalról eltűnik a „Ready” gomb, a szoba elhagyása gomb és a chat ablak megmarad.

A játéktéren a státusz panelt leváltja az aktuális játéktábla.

A képen szöveg, képernyőkép, diagram, Téglalap látható

Automatikusan generált leírás

10. ábra teljes szoba - játék

#### **Játék**

A játék a hagyományos Azul társasjáték szabályait követi. A felhasználók egymás után végzik el a lépéseiket, akcióikat. A soron következő játékos táblája mindig ki lesz emelve, piros színnel, ha az egyik ellenfelünk van soron és zölddel, ha a mi következünk. A saját körünkben választanunk kell egy csempét az egyik piacról (vagy a közös piacról) és a játék táblánk egyik Collected Tiles sorát.

|  |  |
| --- | --- |
| A képen szöveg, képernyőkép, diagram, Téglalap látható  Automatikusan generált leírás  11. ábra csempék kiválasztása | A képen szöveg, képernyőkép, Téglalap látható  Automatikusan generált leírás  12. ábra csempék lehelyezése |

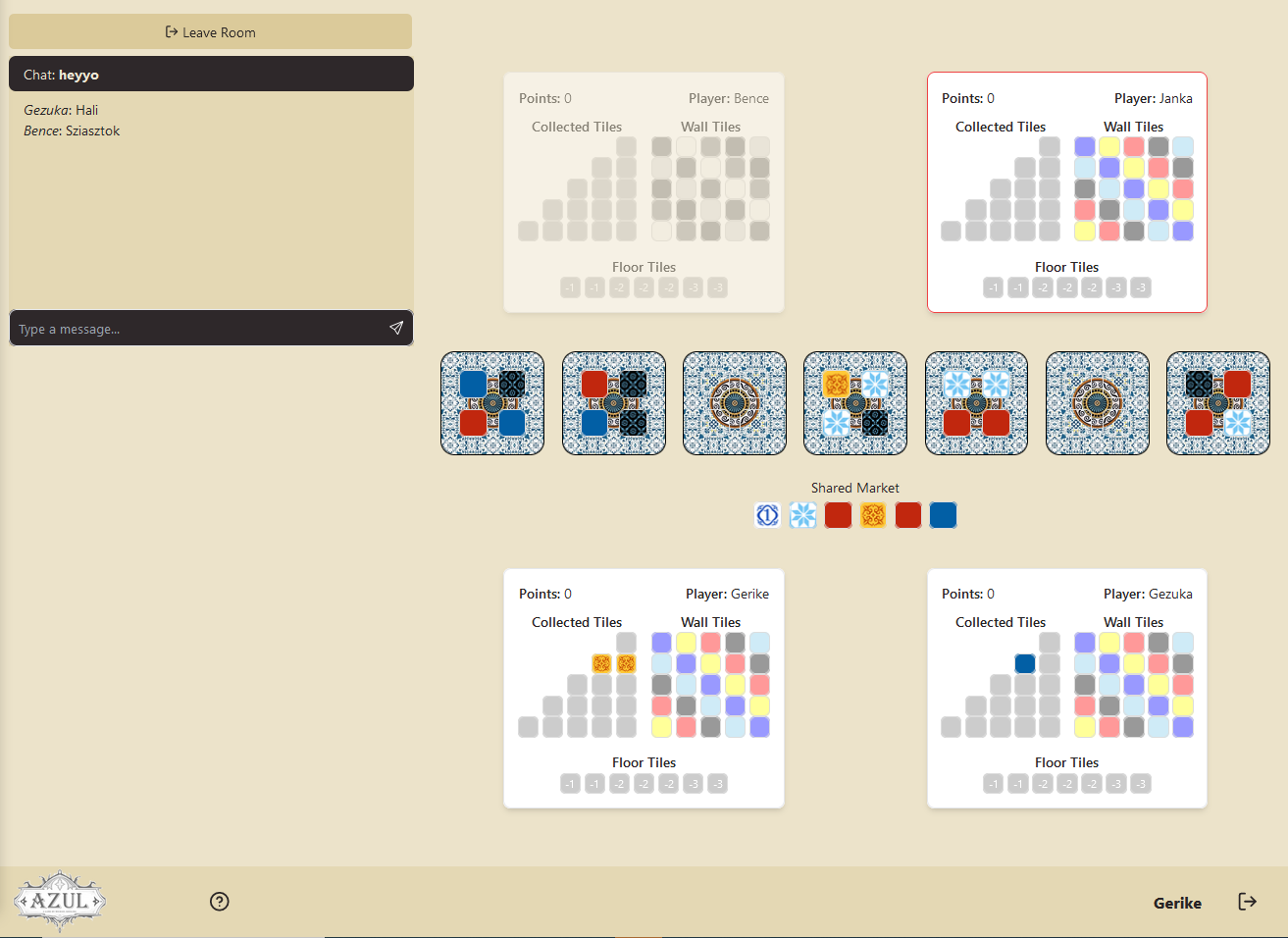
A piacról az összes azonos színű csempét magunkhoz fogjuk venni és a választott sorunkba próbáljuk őket helyezni. Ha a sorban már van csempe, akkor csak vele azonos színűt próbálhatunk a sorba tenni. Ha megtelt a sor, de még nem raktuk le az összes csempénket, akkor a maradék csempe, a földre, a Floor Tiles mezőre fog esni.

|  |  |
| --- | --- |
| 13. ábra Floor példa1 | 14. ábra floor példa2 |

A 13. ábrán kiválasztott 2 fekete csempét a 2. sorba helyezzük, a 14. ábra a lépés eredménye.

Csempét, akkor se helyezhetünk egy sorba, ha az ahhoz a sorhoz tartozó fal soron (Wall Tiles) már van a választott színű csempéből.

Ha egy felhasználó elhagyja játék közben a szobát, akkor az ő tábláját kiszürkítjük és a játék állapotból kitörökjük. A játékmenet úgy fog folytatódni, mintha eleve egyel kevesebben kezdtük volna a játékot. A következő kör elején a piacok számát is újra kalibráljuk.



15. ábra játék egy kilépett játékossal

Miután vége a játéknak a játéktér tartalmát ismét kicseréljük. Most egy játék vége táblát mutatunk a játékosok nevével és az elért pontszámukkal.

*//a játék akkor is véget ér, ha túl sokan lépnek ki és nem tud folytatódni a játék*

A képen szöveg, képernyőkép, Betűtípus, sor látható

Automatikusan generált leírás

16. ábra játék vége panel

Az elfogadásra (“Accept”) gombra kattintva elhagyjuk a játékot és a szobát is. A szoba elhagyásáig a még használhatjuk a közös chat-et.

**2.4.5 Kiskijelzős szoba elrendezés**

A kis kijelzős elrendezésbeen a bal oldali komponenseket elrejtjük és az oldal tetején megjelenítünk egy navigációs gombot, mellyel váltogathatunk a játéktér és a bal panel között.

A képen szöveg, képernyőkép, Számítógépes ikon, szoftver látható

Automatikusan generált leírás

17. ábra kis kijelzős szoba - játék

A játék menet és egyéb funkciók nem változnak.

**2.4.6 Figyelmeztetések**

Az weboldal használata közben az alkalmazás figyelmeztetéseket küld beúszó buborékok formájában.

A képen szöveg, képernyőkép, diagram, Téglalap látható

Automatikusan generált leírás

18. ábra értesítés - rossz lépésről

Ezek csak információs jellegűek. Röviden, néhány szóban beszámol arról, ha a felhasználó valami hibát követett el. Például: Rossz adatokkal próbált meg bejelentkezni/regisztrálni, nem töltött ki egy fontos beviteli mezőt. Játék közben hibás lépést szeretett volna végrehajtani.

# **Fejlesztői dokumentáció**

## **Technológiai Stack**

A szoftver a következő technológiákat használja:

* MongoDB: Dokumentum-orientált NoSQL adatbázis
* Express.js: Minimalista és rugalmas Node.js webalkalmazás keretrendszer
* React: JavaScript könyvtár felhasználói felületek készítéséhez
* Node.js: Szerveroldali JavaScript futtatási környezet
* Socket.io Valós idejű, két irányú kommunikációt tesz lehetővé

## **Rendszer**

A képen szöveg, diagram, vázlat, képernyőkép látható

Automatikusan generált leírás

19. ábra rendszer felépítése

A felhasználó a React App-el, a kliensünkkel kommunikál. A frontend és a web szerverünk http kérés-válasz kommunikációt folytat (bejelentkezés/regisztráció, szobák). A játék és az real-time chat socket kapcsolaton keresztül kommunikál.

## **Kezdeti lépések**

A projekt létrehozásához a következő lépéseket végeztem el:

* Node.js (<https://nodejs.org/>) a legújabb LTS verzió telepítése
* A Projekt Struktúra kialakítása. Létrehozni a root könyvtárat, amiben dolgozni fogunk. Ebben a könyvtárban npm init-el inicializáljuk a Node környezetünket.
* Npm-mel további függőségek telepítése, mint az express, dotenv, cookie-parser, mongoose, bycrptjs, jsonwebtoken, socket.io
* Adatbázis létrehozása a MongoDB Atlas oldalán (<https://www.mongodb.com/cloud/atlas>)
* .env fájl létrehozása kapcsolat és más környezeti változók tárolására. Alkalmazásunknál elég a MongoDB connection string-jét és a PORT-ot ahonnan futtatjuk a szervert belerakni ebbe a fájlba.
* Npm segítségével Frontend (React) létrehozása Vite-tal.
* Frontendhez a socket.io-client telepítése npm-mel. Illetve a további megjelenítési/frontend dependenciák hozzáadása, mint framer-motion, react-dom, react-router-dom, react-hot-toast, react-icons.
* Stílus elemekhez a tailwindcss és daisyui hozzáadása a projekthez.
* Ezekkel a dependenciák, packagek használatával készítettem el a dolgozatomat.
* A felhasznált technológiákról és azok integrációjáról, összekötéséről a következő fejezetekben olvashatnak.

## **Környezet felállítása a további fejlesztéshez**

* a dolgozat fájljainak letöltése
* Node.js telepítése (legfrissebb LTS verzió)
* MongoDB Atlas beállítása, connection string megkeresése

(https://www.mongodb.com/docs/guides/atlas/connection-string/)

* dependenciák telepítése a gyökérkönyvtárban „npm install -y”, a frontend mappában „npm install -y”.
* környezeti változók: a gyökér mappában .env fájl létrehozása. A tartalma:

PORT=<portszám ahonnan fut a szerver>

MONGO\_URI=<Atlas connection string>

SECRET=<egy string, jwt beállításához>

* Környezeti változók: a ./frontend mappában .env fájl létrehozása. A tartalma:

VITE\_APP\_SERVER\_URL=<backend serverunk címe pl: http://localhost:5000>

* server elindítása: a gyökérmappából „npm run server”
* react app elindítása: ./frontend mappából: „npm run dev”

## **Adatbázis (MongoDB)**

A MongoDB egy dokumentum-orientált NoSQL adatbázis, amely JSON-szerű dokumentumokat használ az adatok tárolására. Az alkalmazásomban a MongoDB Atlas-t használtam, amely a MongoDB felhőalapú szolgáltatása. Ez lehetővé teszi az adatbázisok könnyű skálázását, biztonságos tárolását és menedzselését.

**Adatbázis sémák**

Felhasználó:

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

20. ábra User séma

Szoba:

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

21. ábra Room shéma

Üzenethez és a chat-hez

|  |  |
| --- | --- |
| 22. ábra Message séma | 23. ábra Chat séma |

Játék és játékos tábla

|  |  |
| --- | --- |
| A képen szöveg, képernyőkép, dokumentum, szám látható  Automatikusan generált leírás  24. ábra Game séma | A képen szöveg, képernyőkép, Betűtípus, szám látható  Automatikusan generált leírás  25. ábraPlayerBoard séma |

## **Socket.IO Integráció**

A Socket.IO lehetővé teszi a valós idejű, két irányú kommunikációt a szerver és a kliens között. Az alkalmazásban a Socket.IO-t használtuk a valós idejű funkciók megvalósításához, mint például a chat, valós idejű értesítések és a játék eseményeinek küldésére.

A képen szöveg, képernyőkép, dokumentum látható

Automatikusan generált leírás

26. ábra socket.js

A socket eseményekről és azok kezeléséről a Backend és Frontend fejezetekben fogunk részletesebben foglalkozni.

## **Backend (Express.js)**

Technológia

Az Express.js egy minimalista és rugalmas Node.js webalkalmazás keretrendszer, amely számos hasznos funkciót kínál a web- és mobilalkalmazások fejlesztéséhez.

### **Felépítés**

Az alkalmazás szerveroldali kódja az Express.js keretrendszert használja. Az Express app-ot, miután integráltuk a Socket.IO-val, a szerverünkbe importáljuk a különböző útvonalainkkal (route-ok a http kérésekhez) összekötjük, majd a szerver indításakor a MongoDB adatbázishoz is csatlakozunk a connectToMongoDB függvény segítségével, és a szerver futását a megadott porton indítjuk el.

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

27. ábra server.js

Az Express alkalmazáshoz hozzáadjuk a szükséges middleware-eket: express.json() a JSON kérés/test formátum támogatására és cookieParser() a cookie-k kezeléséhez.

### **Szerver-kliens kommunikáció**

**Routes**

Authentikációs és szoba műveleti kérések HTTP kérések formájában érkeznek a szerverre. Az „/api/auth” útvonalra érkező kéréseket az authRouts irányítja tovább, míg az „/api/rooms” útvonalat a roomRoutes kezeli.

Az authentikációs útvonalakhoz az authController biztosít metódusokat a kérés megfelelő kezeléséhez.

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

28. ábra User útvonalak

A roomRoutes útvonalait a roomController-ben kezeljük.

#### **SocketHandlers**

Az alkalmazás azon részei, amelyek valós idejű kommunikációt igényelnek és vagy biztosan több felhasználóval kell kommunikálniuk Socket.IO kapcsolaton keresztül teszik meg azt. A socketHandler-eket a socket integráció alatt kapcsoljuk az apponkhoz.

gameHandler**:**

A játék irányításához szükséges eseményeket figyeli. Ezeket az eseményeket a gameController metódusaival kezeli le a szerver.

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

29. ábra socketHandler a játékhoz

messageController:

a messageControllert használja a sendMessage és getMessage események lereagálásához.

### **Kontrollerek**

Az alkalmazás logikáját tartalmazzák. A megfelelő kezelőkön érkezett adatokat, eseményeket dolgozzák fel, mentik az adatbázisba és továbbítják a frontend felé.

**authController**:

A felhasználóval és adataival dolgozik. Biztosít signup, login, logout és setReady metódusokat a regisztrációhoz, bejelentkezéshez, kijelentkezéshez és a ready státusz megváltoztatásához.

HTTP kéréseket kezelünk le, a kéréshez tartozó adatokat a req (request)ben kapjuk meg. A válaszunkat a res(respons)ban küldjük vissza, res.status-ban jelezve a művelet sikerességét.

* signup: kapott értékeket ellenőrzi, ha még nincs ilyen felhasználó, akkor csinál egy újat. A jelszót bycript segítségével hash-eljük és úgy tároljuk el.
* login: a kapott adatokkal megkeres egy felhasználót és bycript segítségével ellenőrzi a jelszót.
* logout: kiléptetjük a kérést küldő felhasználót. Ha a user szobában van, akkor a szobából is ki kell vegyük, illetve, ha szobában van és megy a játék, akkor a játékból is.

**roomController**:

A szobákon hajt végre módosításokat. A kérések HTTP requestek formájában érkeznek, de a válaszokat, amikor több felhasználóhoz is el kell juttatni, a http válasz mellett Socket.IO eseménnyel is visszaküldi a kliens felé. Metódusai:

* getRooms, visszaküldi a jelenleg hosztolt szobákat. getRoom, az útvonal paraméterében kapott :id alapján keres és visszaküld egy szobát.
* createRoom, létrehoz egy szobát és értesíti róla a klienseket.
* leaveRoom, a küldőt, ha jelenleg szobában van, kilépteti, majd ellenőrzi a szoba tagjainak számát. Ha az nulla, akkor törli a szobát (ha a user játkban volt, akkor kilépteti onnan is).
* deleteRoom: clearUsers-el kiléptet mindenkit a szobából majd kitörli azt.
* senetizeRoom, segéd fv, a szobáról csinál egy jelszó mentes másolatot, amit biztonságosan tudunk elküldeni.

**messageController**:

Csak socket kommunikáció van.

* sendMessage, egy új üzenetet létrehoz az eseményen keresztül kapott adatokból és a felhasználó szobájának címezve elküldi azt.
* getMessages, az eseményen keresztül megkap egy szoba azonosítót, ami alapján megkeres egy chatet és annak összes üzenetét visszaküldi.

**gameController**:

A társasjáték logikáját tartalmazza. Socket kapcsolaton keresztül kommunikál a klienssel.

* setupGame, egy roomId-t kap, amivel keres egy szobát. Ha talált szobát megnézi, hogy van-e hozzátartozó játék. Ha a játék létezik, visszaküldjük azt a szoba tagjainak. Ha nem létezik még játék, akkor újat kezdünk és ezt az új játékot küldjük el a szoba tagjainak(NewGame), majd elkezdünk egy új kört.
* startNewRound, cask a controllerből érhető el. feltölti a piacokat és a közös piacot. majd ezt a jétékállapotot küldi el a szoba felhasználóinak (NewRound)
* takeTiles, kap egy csempét(színt), marketId-t, row-t (sorindex) és a socket handshakeből a userId-t. Ezekből meg tudjuk határozni, hogy melyik játékos, melyik piacról, melyik csempéket vette el. Az elvett csempékkel feltöltjük a játékos táblájának a collectedTiles[row] sorát. Ha szükséges, a floorTiles-t is.TakeTiles eseményt küldönk, majd megnézzük, hogy vége van-e a körnek.
* onRoundOver, csak a controllerből érhető el. Feladata a PlayerBoard-okon a collectedTiles-ból a wallTiles-ba vinni a megfelelő csempéket és ez alapján pontozni a játékosokat. pontozás után megnézzük, hogy vége van-e a játéknak. Ha igen, akkor GameOver eseménnyel küldjük el a játékállást a szoba tagjainak. Ha nem isGameOver, akkkor a RoundOver eseménnyel. Ezután (ha nincs vége a játéknak) startNewRound függvényhívása.
* getGame, egy lekérdezés amely a kapott roomId alapján visszaküld egy játék állapotot (vagy null értket). GetGame eseménnyel küldjük vissza.
* leaveCurrentGame, userId paraméterrel keres egy játékot, amelynek az adott id user a tagja, majd kivesszük a players listából. Ha több, mint 1 játékos maradz, akkor folytatjuk a játékot, PlayerLeftGame eseményt küldünk a játékállásról. Ha 1 vagy 0 játékos maradt a játéban, akkor GameOver eseménnyel lépünk ki.

Segéd fv-ek: newBag, newMarkets, newPlayerBoards, getRandomTileFromBag, loadMarkets, calculatePoints, isGameOver, isRoundOver, isValidMove.

newBag, newMarkets,newPlayerBoards felvesz megflelő mennyiségű üres komponenst a játék állapotba. A loadMarkets a getRandomTileFromBag segítségével feltölti a piacokat csempékkel. Az isGameOver, isRoundOver és isValidMove a játékmenet közben ellenőrzi a játék állapotát, lépés helyességét. A calculatePoints az onRoundOver-ben kiszámolja minden csempe falrahelyezése közben, hogy hány pontot ért.

## **Frontend (React)**

**Technológia**

A React egy JavaScript könyvtár felhasználói felületek készítéséhez. A React lehetővé teszi a komponens alapú fejlesztést, ami egyszerűvé és átláthatóvá teszi a kódot.

**Felépítés**

**Context-ek**

Context-ek adatközlésre szolgálnak a ReactDOM fában. Használatával elkerülhetjük a szülő-gyerek elemek közti property alapú adatátadást.

A dolgozatban az App komponens köré vettem fel a kontextusaimat, így azokat és adatait/funkcióit az összes komponensből elérjük.

A képen szöveg, képernyőkép, Betűtípus, dokumentum látható

Automatikusan generált leírás

30. ábra main.jsx

Az ábrán látható main.jsx a React alkalmazásunk belépési pontja. Az index.html body tag-jében szereplő <div id=”roor”></div> elemben hozza létre az App.jsx komponensünket. A main.jsx-ben található ContextProvider-ek felelnek a gyerekelemek felé történő adatátadásért.

**AuthContex:**

A jelenleg bejelentkezett felhasználót tárolja el egy useState hookban és a localstorage-ben. Biztosít egy useAuthContext funkciót, amivel megtudjuk hívni a kontextust és a visszatérési értéke a provider melynek value paraméterében tovább adjuk az authUser változót és a setAuthUser setter metódust.

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

31. ábra AuthContext

**SocketContext:**

a socket.io-client ből importált io segítségével létesít socket kapcsolatot a szerverrel. Majd ehhez a socket-hez biztosít hozzáférést az App-ban.

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

32. ábra SocketContext részlet

**RoomContext:**

A jelenleg aktív szobákat tároljuk benne. Biztosít egy rooms változót és egy setRooms setter metódust. const [rooms, setRooms] = useState([]);

**ChatContext:**

Ennek a kontextusnak (Provider-nek) a használatával elérhetővé tesszük a messages állapot változót és a setMessages setter metódusát.

**GameContext:**

A játékállapotot tároljuk benne egy gameState állapot változóban és biztosít számunkra egy setGameState setter metódust.

**Saját hookok**

**useSignup:**

Regisztrációhoz használjuk. Biztosít számunkra egy signup függvényt, ami egy http kérést küld a szervernek. Sikeres válasznál beállítjuk az authContext authUser-ét.

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

33. ábra useSignup hook

**useLogin:**

Bejelentkezéshez használjuk. Biztosít login metódust, ami http kérést küld a szervernek és a válasz alapján beállítja az authUser-t.

**useLogout:**

Biztosít egy logout metódust, ami egy http kérést küld a szervernek és beállítja az authUsert null értékre, majd törli a localstorage-ből.

**useJoinRoom:**A következő metódusokat tartalmazza: joinRoom, leaveRoom, getRooms. Ezekkel http kéréseket küldünk a szervernek a szobák kezelésével kapcsolatban. joinRoom-mal csatlakozunk egy meglévő szobához. leaveRoom-mal kilépünk a jelenlegi szobából. A getRooms-szal lekérdezhetjük az összes akítv szobát. Ha sikeres értékkel tér vissza egy request, akkor frissíti az authUser-t és a rooms-ot (roomContext-ből) is.

**useListenRoom:**

A szobákhoz tartozó socket eventeket figyeljük rajta. Ezek az eventek a következők. newRoom, hozzáadja az eseménnyel érkező új szobát a rooms értékéhez. deleteRoom, kitörli az eventtel érkező szobát a rooms-ból. updateRoom, egy rooms-beli szobát frissít. getRooms az egész rooms értéket felülírja az eventből kapott új szobákkal.

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

34. ábra useListenRooms

**useGame:**Socket eseményeket küld a szervernek.

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

35. ábra useGame

setupGame, takeTiles, getGame metódusokat biztosít a számunkra. setupGame-mel megpróbálunk egy új játékot kezdeni. A takeTiles metódussal irányítjuk a játékot. A getGame lekérdezi a jelenlegi játékállást.

**useListenGame:**

A szerver felől érkező, a játékhoz kapcsolódó socket eseményeket kezeli. Ezekkel az eseményekkel frissítjük a gameState megfelelő adattagjait, illetve bizonyos esetekben meganimáljuk a játék változásait. Hogy biztosítsuk, a megfelelő sorrendű animációt és követhető legyen a játék változása egy sorba tesszük a beérkező eseményeket és egyesével kezeljük le őket.

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

36. ábra useListenGame eventQueue

Ezek az események a backend/cotrollers/gameController-éből érkeznek.

**useListenMessages:**A chat funkcionalitásához szükséges. Új üzeneteket fogad, melyek a newMessage socket eseménnyel érkeznek. Amikor szobát váltunk vagy frissül a *socket* (socketContextből) egy socket eseménnyel, a getMesseges-el lekérdezzük a szobához tartozó jelenlegi üzeneteket. Ugyan ezen, a getMessages néven kapjuk vissza a szoba összes eddigi üzenetét. Továbbá ez a hook biztosít egy sendMessage metódust új üzenet küldésére.

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

37. ábra useListenMessages

App

A React alkalmazás React Router DOM-ot használ a kliens oldali útvonalkezeléshez, lehetővé téve az egyoldalas alkalmazás (SPA) élményt.

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

38. ábra App

**További frontend komponenseink**

**Signup oldal**

Egy űrlapot ábrázol mellyel megadható a felhasználó neve, jelszava és a jelaszava még egyszer. Ezekkel az adatokkal indítja meg a regisztrációs folyamatot. Ha sikeres átirányít a Home oldalra.

**Login oldal**

Egy űrlapot ábrázol felhasználónévvel és jelszóval. Ezekkel az adatokkal megpróbál bejelentkezni, majd a Home oldalra irányít.

**Home page**

A RoomForm és HomeSideBar-t jeleníti meg.

Feltölti a roomsContext-et a useJoinRoom::getRooms metódussal és a HomeSideBar-nak paraméterként odaadja a rooms-ot.

**RoomForm**

Egy űrlap szoba készítéshez. Az új szobára automatikusan meghívja a useJoinRoom::joinRoom metódusát.

**HomeSideBar:**

Megjelenít egy keresési inputot és a jelenlegi szobákat kártya formában. A kapott rooms propból minden szobára csinál egy RoomCard-ot. A keresési input leszűri a RoomCard-ok listáját név alapján.

**RoomCard:**

Propertynek kap egy szobát és kártya formában jeleníti meg. Ha a szobának van jelszava, akkor ad a kártyához egy jelszó beviteli mezőt. A csatlakozás gomb onClick eseményéhez hozzárendeli a useJoinRoom::joinRoom metódusát.

**Session oldal:**

Tartalmazza a ChatBox komponenst, gombokat a játékos státusz váltásához és a szoba elhagyásához. Illetve megjeleníti a játékteret, ami a jelenlegi játék állapot és isGameOver state változó függvényében a következő lehet: StatusBoard, GameOverPanel vagy maga a játék tábla. A játéktábla PlayerBoardCard-okból, Market-ekből és SharedMarketből áll.

A szoba elhagyását a useJoinRoom::leaveRoom metódusa végzi a „leave button” onClick eseményén. A játékos státusz váltását a handleToggleReady végzi.

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

39. ábra handleToggleReady

Ez a függvény módosítja a Session komponens PlayerReady state változóját.

Az alábbi useEffect-tel figyeljük, hogy el tudjuk-e indítani a játékot:

A képen szöveg, Betűtípus, sor, képernyőkép látható

Automatikusan generált leírás

40. ábra start game useEffect

a canStartGame igazzal tér vissza, ha megfelelő számú felhasználó van a szobában és mindenkinek „ready” státusza van. a handleSetupGame a useGame::setupGame segítségével indítja a játékot.

A StatusBoard-ot csak akkor mutatjuk, ha még nem indult el a játék és a gameOver state változó hamis.

{!gameState && !gameOver && (

    <StatusBoard />

A játéktábla elemeit a gameState-ből (GameContext) töltjük fel, illetve a játék irányításához függvényeket adunk át property-k ként a játék komponenseknek. A Market és SharedMarket az handleTileClicked metódust kapják meg, amivel a selectedTile és selectedMarket state változóknak adnak értéket.

A képen szöveg, Betűtípus, sor, képernyőkép látható

Automatikusan generált leírás

41. ábra csempe kiválasztása

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

42. ábra csempe választás átadása

A PlayerBoardCard megkapja a handleRowClick metódust, mellyel a selectedRow state változó kap értéket.

Miután ez a három változó kapott értéket meghívjuk a handleTakeTile metódust, ami a useGame::takeTiles fügvénnyel elküldi socket-en keresztül a szerver felé a játék akciót.

A képen szöveg, Betűtípus, sor, képernyőkép látható

Automatikusan generált leírás

43. ábra takeTiles

A játék a Session.jsx oldalon addig tart, amíg nem érkezik GameOver socket esemény a szerver felől, ekkor a gameOver értéket igazra váltjuk, ezzel feltételesen rendereljük a GameOverPanel-t.

A képen szöveg, képernyőkép, Betűtípus, sor látható

Automatikusan generált leírás

44. ábra GameOverPanel

**StatusBoard:**

Property-ként megkapja az adott szoba users (felhasználói) tömbjét és táblázatos formában mutatja a játékosok nevét és státuszát.

**GameOverPanel:**

Property-ként megkapja a játék játékostábláinak tömbjét, a játékosnév és szerzett pont adatokból csináltáblázatot. Megjelenít egy ”Accept” gombot, melynek az onClick eseménye meghívja a useJoinRoom::leaveRoom metódusát

**Tile**

Property-ként kap egy tile-t a játék állapotból, egy onClick eseményt (Session-ből kapta a Market és továbbadja a Tile-nak), isSelected (property jelzi, ha ki van jelölve a Tile) és egy id-t, ami fontos lesz az animációk miatt.

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

45. ábra Tile

**Market:**

A piac csempéket (Tile) tart. Ezeket a gameState-től a tiles property-én keresztül kapja meg.

onTileClick , selectedTile, selectedMarket property-eit továnnküldi a gyerek(Tile)einek

**SharedMarket:**

Hasaló a Markethez, itt a marketId fix -1

**PlayerBoardCard:**

propertynek kap egy playerBoardot és egy onClick metódust. A playerBoarddal feltölti adatokkal a kártyát és a collectedTiles soraihoz hozzárendeli az onClicket. Ez szükséges lesz a csempe mozgatásához (Session.jsxben handleTakeTile)

A kártya tartalmazza a játékos nevét, pontszámát, collectedTiles-,wallTiles- és florrTiles tömbjét.

**ChatBox:**

Egy chat ablakot ábrázol. Két komponense a Messages és a MessageInput. Ezen felül mutatja a szoba nevét.

**Messages:**

A useListenMessages::messages értékével készít listát Message komponensekből. A legutóbbi üzenetet a React useRef hook segítségével eltároljuk és egy useEffect hook-kal legörgetünk hozzá. Így mindig látni fogjuk a legfrissebb üzenetet.

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

46. ábra Messages

**MessageInput:**

Egy form, amely egy input mezőből és egy küldés gombból áll. A submit eseményt a gombbal és az enter billentyű lenyomásával is kiválthatjuk. Egy useState változóban message tároljuk az aktuális üzenetet és a useListenMessages::sendMessage metódusával küldjük el.

**Message:**

Property-nek kapott message-ből készít egy megjeleníthető üzenetet.

return (

        <div className={`message ${authUser.\_id === message.senderId ? "sent" : "received"}`}>

            <p><i>{message.senderName}</i>: {message.message}</p>

        </div>

    );

**Navbar:**

Egy navigációs sáv, ami tartalmazza a kijelentkezéshez szükséges gombot, egy súgó gombot és megjeleníti a bejelentkezett felhasználó nevét. A Navbar tartalmazza a HelpPanel komponenst is, amelyet egy showHelpPanel useState változó értéke alapján mutatunk a felhasználók felé. A showHelpPanel értékét a súgó gomb onClick eseményével változtatjuk.

**HelpPanel:**

Súgóként használjuk, információt biztosít az alkalmazás használatáról HowToUse komponens és a játék szabályairól GameRules komponens. Hogy melyik komponenst kell megjeleníteni azt a [selectedTab, setSelectedTab] = useState változóban tároljuk. selectedTab értékét a HelpPanel „Game Rules” és „How to use” gombjaival tudjuk beállítani. A súgó tartalmaz még egy gombot az ablak bezárásához.

**HowToUse, GameRules:**

Statikus komponensek, egy JSX állománnyal térnek vissza.

**Animációk**

Az AnimateChanges.js fájl biztosít számunkra egyszerű animációkat a játékhoz. Ezeket a useListenGame-ben használjuk, amikor kapunk egy új játék állapotot. Összehasonlítjuk a mostani és a következő gameState-et és az alapján végezzük el az animációkat a következő metódusokkal:

* animateTakeTiles, csempék mozgatása a TakeTiles esemény alapján Market -> PB (PlayerBoard)
* animateRoundOver, csempék mozgatása RoundOver esemény alapján PB.collectedTiles -> PB.WallTiles

Használunk egy handleTileMove segéd metódust, ami kap egy toElementId-t és egy fromElementId-t. DOM-on megkeresi ezeket és a két pont között egy ideiglenes csempe mozgását animálja meg.

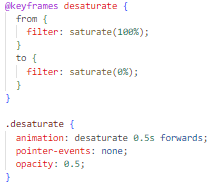
A képen szöveg, képernyőkép, Betűtípus, dokumentum látható

Automatikusan generált leírás

47. ábra Csempe mozgatás animáció

* animatePlayerLeft, PlayerLeftGame eseménynél kiszürkíti a kilépett játékos tábláját, ezzel jelezve, hogy a játékos elhagyta a játszmát. Ehhez az átmenethez egy CSS stílust adunk a játékos táblájához

leftPlayerBoardElement.classList.add('desaturate');



48. ábra kiszürkítős css

**Tesztelés**

A teszteléshez egy új adatbázist hozok létre. Ehhez elegendő a MONGO\_URI környezeti változóban tárolt connection stringet átírni. Ha megváltoztatjuk a DB nevét, akkor a MongoDB Atlas egy új Collection-t hoz létre az új névvel.

A tezteléshez egy feketedoboz megközelítést alkalmazok, ahol a webalkalmazás funkcióit fogom kézzel tesztelni, közben ellenőrizve azok helyes működését. A teszteseteket az alkalmazás funkciói alapján csoportosítottam, leírtam, hogy az adott funkciót miként használtam és mit tapasztaltam.

**Regisztráció**

* Nem töltünk ki minden mezőt: el se küldjük a http requset-et, hibát kapunk. Toaster error-ral van kezelve.
* Kitöltünk minden mezőt, de nem azonos jelszót adunk meg: status 400 Bad request, toaster erroral jelezve (Passwords don’t match).
* Jelszavak egyeznek, de tól rövid: status 400 Bad request, toaster error(Password must be at least 6 long).
* Helyes bejelentkezés: beléptünk a fő oldalra. User adatai megjelentek az adatbázisban, authUser eltárolva a localstorage-ben.
* Újra regisztrálás azonos névvel: status 400 Bad request, toaster error(username already exists).

**Bejelentkezés:**

* Nem töltünk ki minden mezőt: el se küldjük a http requset-et, hibát kapunk. Toaster error-ral van kezelve.
* Helytelen (nem létező username) névvel kitöltve: status 400 Bad request, toaster error(Invalid credentials).
* Helyes username, rossz jelszó: status 400 Bad request, toaster error(Invalid credentials).
* Helyes adatok megadása: beléptünk a fóoldalra. authUser eltárolva a localstorage-ben.

**Kijelentkezés:**

* Kijelentkezés a webfelületről: helyesen kijelentkeztünk, localstorage authUser-e null, átirányított minket a Login page-re.
* Nem vagyunk bejelentkezve: ReactDOM-ban nincs is felvéve a logout elem.
* Szobába vagyunk, játék nélkül: sikeres kijelentkezés, localstorage authUser-e null, minket átirányított a Login page-re.
* Szobában vagyunk, fut a játék: sikeres kijelentkezés, localstorage authUser-e null, minket átirányított a Login page-re.

**Fő oldal / Szobák:**

* Névnélküli szoba létrehozása: status 400 Bad request, toaster error (couldnt host room)
* Szoba hosztolása csak név: szoba felvételre került, felhaználó automatikusan csatlakozott, átnavigált a szoba nézetre.
* Szoba hosztolása egy meglévő szoba nevével: status 400 Bad request, toaster error (couldnt host room)
* Meglévő várokozó szobához csatlakozni, ahol < 4 felhasználó van: a felhasználó sikeresen csatlakozott, átnavigált a szoba nézetre.
* Meglévő szobához csatlakozni, ahol 4 felhasználó van: nem lehet csatlakozni, nincs is rá lehetőség.
* Teli vagy játékot már megkezdett szobához csatlakozni: nincs lehetőségünk.

**Chat:**

* Üzenet küldés: a szoba összes felhasználójánál megjelenik.
* Meglévő szobához csatlakozva: ha volt beszélgetés, akkor az töltődik be. Ha nem volt, akkor üres.
* Más szobához csatlakozva: előző beszélgetésünk helyett, az új szoba beszélgetését látjuk.

**Szoba elhagyás:**

* Nem utolsóként hagyjuk el: visszakerülünk a fő oldalra, a szoba még a listában.
* Utolsóként hagyjuk el: visszakerülünk a fő oldalra, szoba kitörlődik a listából.
* Elhagyás utáni újra csatlakozás: kiszámítható módon a fő oldal esetei szerint.

**Játék:**

* Ready-zés, mindenki ready: A játék elindul, szoba állapota „game in progress”-re vált. Ready button eltűnik.
* Ready-zés, nem mindenki ready: kiszámíthatóan csak akkor indul el, ha egy pillanatban minden felhasználó ready.
* Nem a saját körünkben lépünk: nem történik semmi, toaster error(Not your turn)
* Sajátkörünkben nem szabályos lépé: nem történik semmi, toaster error(Invalid move)
* Saját körünkben szabályos lépés: Csempék játékszabály szerint kerülnek elhelyezésre, mindenkinél frissül a játékállapot. Új játékos következik.
* A kiválasztott csempénket az előző játékos kihúzza, majd a sorunkra kattintunk: nem történik semmi, továbbra is a mi körünk van. toaster error(No tiles to take)
* Egy felhasználó játék közben kilép: játék megy tovább, a kilépett játékos táblája kiszürkül, sose fog sorra kerülni. Következő körtől a piacok száma újra csökken.
* Egy felhasználó saját köre közben kilép: kiszürkül a táblája, a következő játékos kerül sorra. A játék zökkenőmentesen folytatódik.
* Egy felhasználó kilépése után egyedül maradunk játékban: azonnali Game Over, nem tudjuk folytatni a játékot. Játék vége panel megjelenítése.
* A játékszabályok szerinti játék vége: Játék vége panel megjelenítése.
* Játék vége elfogadása: szoba elhagyása gombbal egyenértékű.

A tesztelés célja a felhasználók számára egy kiszámítható, megbízható és hibamentes weboldal biztosítása. A manuális tesztelés során minden lehetséges felhasználói interakciót kipróbálunk, hogy az esetleges hibákat és hiányosságokat azonosítsuk. A cél egy olyan felhasználói élmény biztosítása, amely minden elvárásnak megfelel, és zavartalan felhasználói élményt biztosít.

A program jelenleg minden(fenti) tesztnek eleget tesz. A tesztelés alatt egy hibát találtam, a „A kiválasztott csempénket az előző játékos kihúzza”. Ezt azonnal javítottam, a játék alatt nem tudunk már nem létező csempét kihúzni.

# **Összefoglalás és további fejlesztési lehetőségek**

## **Összefoglalás**

Ez az alkalmazás, melyet a hozzám hasonlóan társasjáték-kedvelő célközönség számára fejlesztettem, egy online platformot biztosít, ahol barátok, családtagok, ismerősök vagy akár teljesen idegenek közösen élvezhetik a társasjátékot, függetlenül a fizikai távolságtól. Az alkalmazás tervezése és fejlesztése során kiemelten fontos szempont volt, hogy a felhasználói élmény egyszerű és intuitív legyen. Ennek érdekében a felhasználói interfészt úgy alakítottam ki, hogy az magától értetődő legyen, és bárki könnyen tudja használni, akár technikai háttérismeret nélkül is.

Az alkalmazás tartalmazza a játékok indítását, csatlakozást a különböző játék szobákhoz, és a játék közbeni kommunikációt, ami lehetővé teszi a valós idejű interakciót a játékosok között.

## **További fejlesztési lehetőségek**

Továbbfejlesztéshez az első ötletem az lenne, hogy a jelenlegi 1 társasjátékot implementáló weboldalból egy társasjáték hub-ra fejlesztenénk. Az oldal szerkezetét csak részben kéne módosítani. A szoba készítésnél lehetőséget adnánk társasjáték választásra és az így létrejövő szoba felhasználóinak száma a játékéhoz azonosulna. A jelenlegi Session page-ből absztraktálnánk a játékot, helyette különböző játék komponenseket készítenénk. Minden egyes új játékhoz továbbá szükségünk lesz:

* Egyedi hookokra, hogy tudjunk a szerverrel kommunikálni
* Adatbázis sémára a játékhoz
* Socket kezelőre
* új játék kontrollerre

# **Irodalomjegyzék, hivatkozások**

1. Azul:<https://planbgames.com/next-move/games/azul-next-move-games-michael-kiesling-strategy-abstract-board-game-winner-spiel-des-jahres-game-of-the-year-cannes-portuguese-tiles-royal-palace-of-evora-5>

(A játék hivatalos kiadója)

1. Node.Js: <https://nodejs.org/en>
2. ExpressJS: <https://expressjs.com/>
3. MongoDB: <https://www.mongodb.com/>
4. MongoDB Atlas: <https://www.mongodb.com/products/platform/atlas-database>
5. React: <https://react.dev/>
6. Tailwind CSS: <https://tailwindcss.com/>
7. DaisyUI: <https://daisyui.com/>
8. Socket.IO: <https://socket.io/>