Eötvös Loránd Tudományegyetem

Informatikai Kar

Programozáselmélet és Szoftvertechnológiai Tanszék

Körökre osztott társasjátékokat megvalósító webalkalmazás

Szerző:

Wolosz András

programtervező informatikus BSc

Belső témavezető:

Nagy Sára

Mesteroktató

Külső témavezető:

Vadász Péter

MSc

Budapest, 2021

EÖTVÖS LORÁND TUDOMÁNYEGYETEM
INFORMATIKAI KAR

SZAKDOLGOZAT TÉMABEJELENTŐ

Hallgató adatai:
Név: Wolosz András
Neptun kód: PQINMC

Képzési adatok:
Szak: programtervező informatikus, alapképzés (BA/BSc)
Tagozat : Nappali

Külső témavezetővel rendelkezem

Külső témavezető neve: *Vadász Péter*
munkahelyének neve: ***Systemathx Hungary Zrt.***
munkahelyének címe: ***1118 Budapest, Menta u. 4.***
beosztás és iskolai végzettsége: ***Fullstack webfejlesztő (végzettség: Programtervező informatikus MSc)***
e-mail címe: ***pevad95@inf.elte.hu***

Belső konzulens neve: *Nagy Sára*
munkahelyének neve, tanszéke: ***ELTE-IK, Algoritmusok és Alkalmazásaik Tanszék***
munkahelyének címe: ***1117, Budapest, Pázmány Péter sétány 1/C.***
beosztás és iskolai végzettsége: ***Mesteroktató***

A szakdolgozat címe: Körökre osztott társasjátékokat megvalósító webalkalmazás

A szakdolgozat témája:
(A témavezetővel konzultálva adja meg 1/2 - 1 oldal terjedelemben szakdolgozat témájának leírását)

A cél egy olyan webalkalmazás elkészítése, mellyel körökre osztott társasjátékokat lehet játszani, valamint ezekkel kapcsolatos eredményeket kezelni. Az alkalmazás lehetőséget biztosít valós idejű játékra (más felhasználók ellen). Különféle játékok közül lehet választani, elérhető ezek leírása, illetve a kapcsolódó ranglista. A regisztrált felhasználók számára lehetőség van játék indításra, játékelőzmények megtekintésére, korábbi játszmák visszatekintésére. A korábbi játékokhoz megjegyzések is fűzhetők. A felhasználóhoz a játszott játékok és elért eredmények alapján egy szintszám rendelődik, amely folyamatosan növelhető. Lehetőség van különféle "teljesítmények" (achievement) elérésére is.

A szakdolgozat REST API szerű programként épül fel. Az alkalmazás front-end, illetve back-end programokat foglal magába. A front-end rész HTML, JavaScript (Angular) alapú, a backend PHP.

Budapest, 2021.04.20.

Tartalomjegyzék

1. fejezet

Bevezetés

> „Játék? Lehet. De mi lesz belőle, ha komolyan vesszük? Nemsokára fölösleges lesz felkeresni egymást, földlakók, hogy beszéljünk. Érintkezni, személyesen (mint a szó is mutatja) csak azok számára jelenti a másik élőlény fizikai jelenlétének szükségességét, akiknek apróbb dolguk is van egymással, mint hogy lássanak és halljanak [...]"
>
> Karinthy Frigyes: *Játék, ajándék.* 1930.

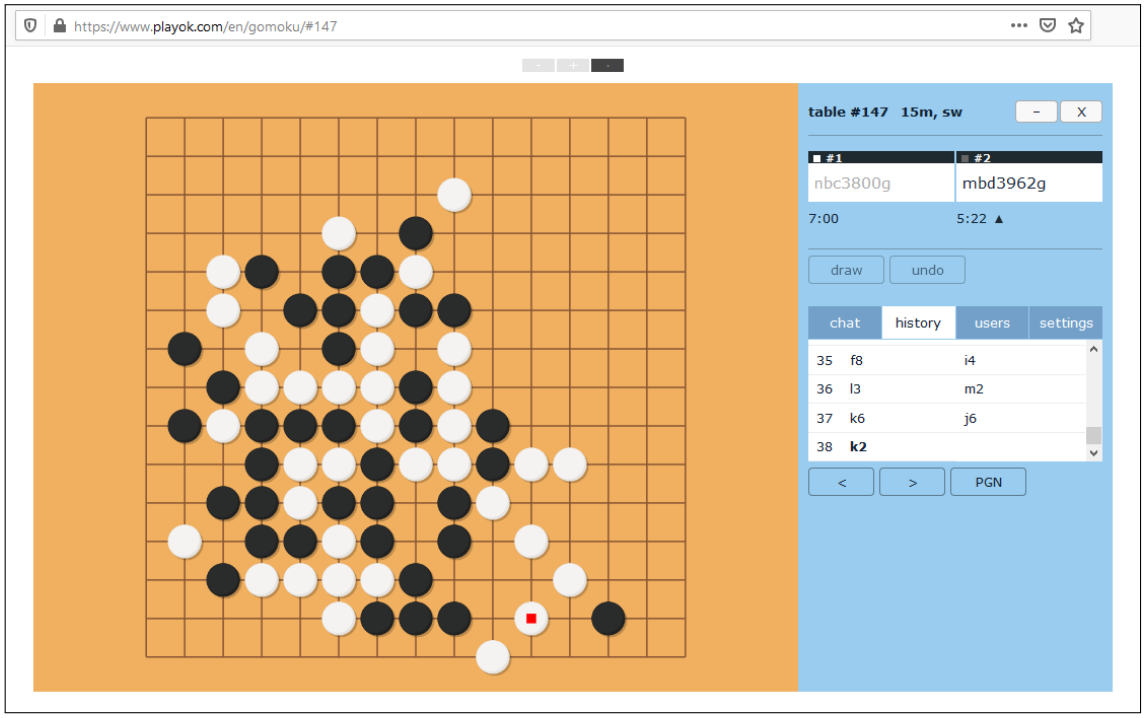
Karinthy Frigyes a *Játék, ajándék* című novellájában mondhatni megjövendöli az online távkommunikációt. Napjainkban az internetes kapcsolattartás már nemcsak egy lehetőség, hanem az életünkhöz tartozó szükség, ahol az oktatás és a munka is ezen a csatornán folyik. Ugyanígy, az életünkhöz tartozó szükség a szórakozás is.

1.1. Témaválasztás

Napjainkban a társasjátékok egyre nagyobb népszerűségnek örvendenek. A nagy lelkesedésnek hála, a kiadók egyre több játékot tudnak a piacra dobni, egyre komplexebb vagy egyre frappánsabb játékmechanikákkal. A feltörekvő piaci ágat az informatika is felismerte, és egyre több társasjátéknak készül el az online is játszható verziója, ezzel utat nyitva az egymástól távol lévő barátok számára.

Némely alkalmazások azonban nagyobb távlatokban gondolkodnak. Nem csupán egy, hanem több játékot is implementálnak ugyanarra a platformra. Ezek a legfőbb előnyei, hogy egy felhasználó több játékot is kipróbálhat, strukturáltan férhet hozzá

a játékokkal kapcsolatos információkhoz. A felhasználó végül egy játékokon átívelő platformot kaphat. A legnagyobb ilyen weblap a *Board Game Arena*, az asztali alkalmazások közül pedig a *Tabletop Simulator*. De gondolhatunk különböző sakkoldalakra, vagy a *Litle Golem*re, ahol akár játékbotok fejlesztői is összemérhetik erejüket.[1]



1.1. ábra. Gomoku játszma a playok.com-on

Szakdolgozatomban egy ilyen weboldal megteremtése volt a célom. A **Sánc** egy webes alkalmazás, ahol körökre osztott játékokkal tudnak játszani a regisztrált felhasználók. Az oldalon lehetőség nyílik a megkezdett játékok visszanézésére, újrajátszására stb. Elérhető admin funkció is, ahol az adminoknak játékokra és a felhasználókra egyaránt teljeskörű rálátása adódik.

[1] en.boardgamearena.com
www.tabletopsimulator.com
www.littlegolem.net
www.playok.com [Hozzáférés valamennyihez: 2021. április 19.]
A linkekre mutató lábjegyzetek kattinthatóak a részletekért.

1.2. Alkalmazott eszközök

Az alkalmazott keretrendszereket elsősorban az alapján választottam, hogy mennyire népszerűek ma a webes területeken. Fontos volt számomra, hogy a későbbiekben akár a munkaerőpiacon is tudjam kamatoztatni a dolgozat elkészítése közben elsajátított ismereteket. Ezen kívül ezekkel a rendszerekkel már egyetemi tanulmányim során megismerkedtem – így könyebben haladhattam a projekttel, mint ha az alapoktól kezdve kellett volna megtanulnom új eszközök használatát.

2. fejezet

Felhasználói dokumentáció

2.1. Sánc

A **Sánc** különböző körökre osztott stratégiai játékok felülete. Az oldalra a fejlesztő által implementálhatóak játékok. Egy felhasználó vendégként, regisztrált felhasználóként vagy adminként navigálhat az oldalon. A játékok újra megtekinthetőek, visszajátszhatóak, az eredmények elérhetőek, hozzájuk megjegyzés fűzhető. A játékokhoz játékszabályok és ranglisták is tartoznak.

Lehetőség van szintlépésre, és achivementek megszerzésére. Az admin széleskörűen hozzáfér a felhasználók és a szobák adataihoz, illetve van törlési jogosultsága. Jelenleg két sokak által ismert játék érhető el az oldalon, a gomoku és a negyedelő.

A **Sánc** alapvetően mindenki számára érdekes lehet, aki érdeklődik a társasjátékok iránt, használta nem igényel informatikai ismereteket.

A weboldal nincs kihelyezve stabil webszerverre, így csupán lokálisan érhető el. A stabil webszerver hiányának okairól bővebb információk a 4.1 fejezetben olvashatók.

2.2. Rendszerkövetelmények

Böngészők

- Firefox - `v84.0`
- Chrome - `v88.0`

Lokális szerverkörnyezetek

- LAMP (Linuxra)
- WAMP (Windowsra)

Egyéb

- Composer - `v2.0.9`
- Node.js - `v14.15.4`
- npm - `v6.14.10`
- PHP - `v7.3`
- Angular CLI - `v11.1`

2.3. Telepítés

A telepítéshez navigáljunk el a `sanc` elnevezésű gyökérmappába. A `client` almappában adjuk ki a következő parancsot:

```
$ npm i
```

2.1. kód. Kliensoldal telepítése

Ez fogja letölteni a kliensoldali kód futtatásához szükséges egyéb függőségeket. A szerveroldali függőségek behúzásához lépjünk vissza a `server` almappába. Futassuk az alábbi utasítást:

```
$ composer i
```

2.2. kód. Szerveroldal telepítése

Az alkalmazás telepítése sikeresen megtörtént.

2.4. Futtatás

A `server` mappában maradva a szerver elindításához az alábbi parancsra van szükségünk:

```
$ php artisan serve
```

2.3. kód. Szerveroldal futtatása

A szerver a localhost:8000-es porton figyel. A `client` mappában futtassuk a következő kódot:

```
$ ng s -o
```

2.4. kód. Kliensoldal futtatása

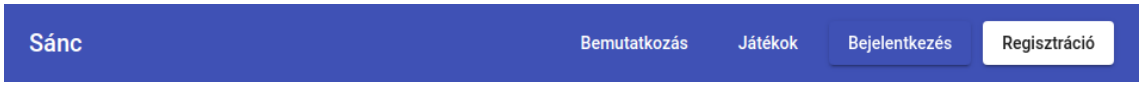
Kis várakozás után felugrik a böngészőnk a localhost:4200-as porton. Az alkalmazásunk kipróbálásra kész.

2.5. Program ismertetése

2.5.1. Navigációs menü

Az oldal tetején található a navigációs menü. Ennek jobb oldalán sorakoznak azok a gombok, amelyek átirányítanak más aloldalakra.

A navigációs menü megváltozik, ha bejelentkezett felhasználó használja az oldalt.



(a) Bejelentkezés előtt

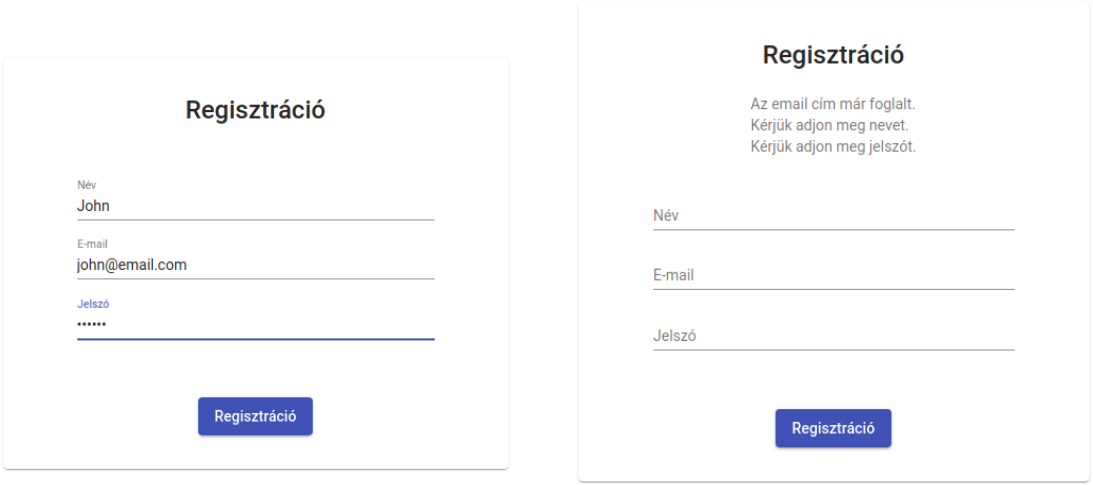
(b) Bejelentkezés után

2.1. ábra. Navigációs menü

2.5.2. Regisztráció

A menüben a Regisztráció gombra kattintva nyithatjuk meg a regisztrációs komponenst. Ezt helyesen kitöltve regisztrálhatunk a **Sánc**ra. Az oldal automatikusan továbbirányít a nyitólapra. Ezt követően tudunk bejelentkezni a Bejelentkezés fülön.

A komponens egy üzenettel jelzi, ha valamilyen validációs hibát észlel. Validációs hiba lehet, ha üresen marad valamelyik mező, vagy az email cím nem felel meg a formai követelményeknek, esetleg ha már létezik felhasználó az adatbázisban ugyanezzel az email címmel.



(a) Helyesen kitöltött (b) Hibásan kitöltött

2.2. ábra. Regisztrációs komponens

2.5.3. Bejelentkezés

A belépés komponensre navigálva van lehetőségünk belépni az alkalmazásba. A bejelentkezés oldal a helyesen kitöltött mezőkkel irányít tovább a profil oldalra.

Ha hibásan töljtük ki, vagy üresen hagyjuk valamelyik mezőt, akkor a regisztrációs hibához hasonló hibaüzenetet kapunk.

Belépés után megváltozik a navigációs menü. Megjelennek rajta a Profil és a Kijelentkezés gombok, eltűnnek a Regisztráció és a Belépés gombok.

2.5.4. Profil oldal

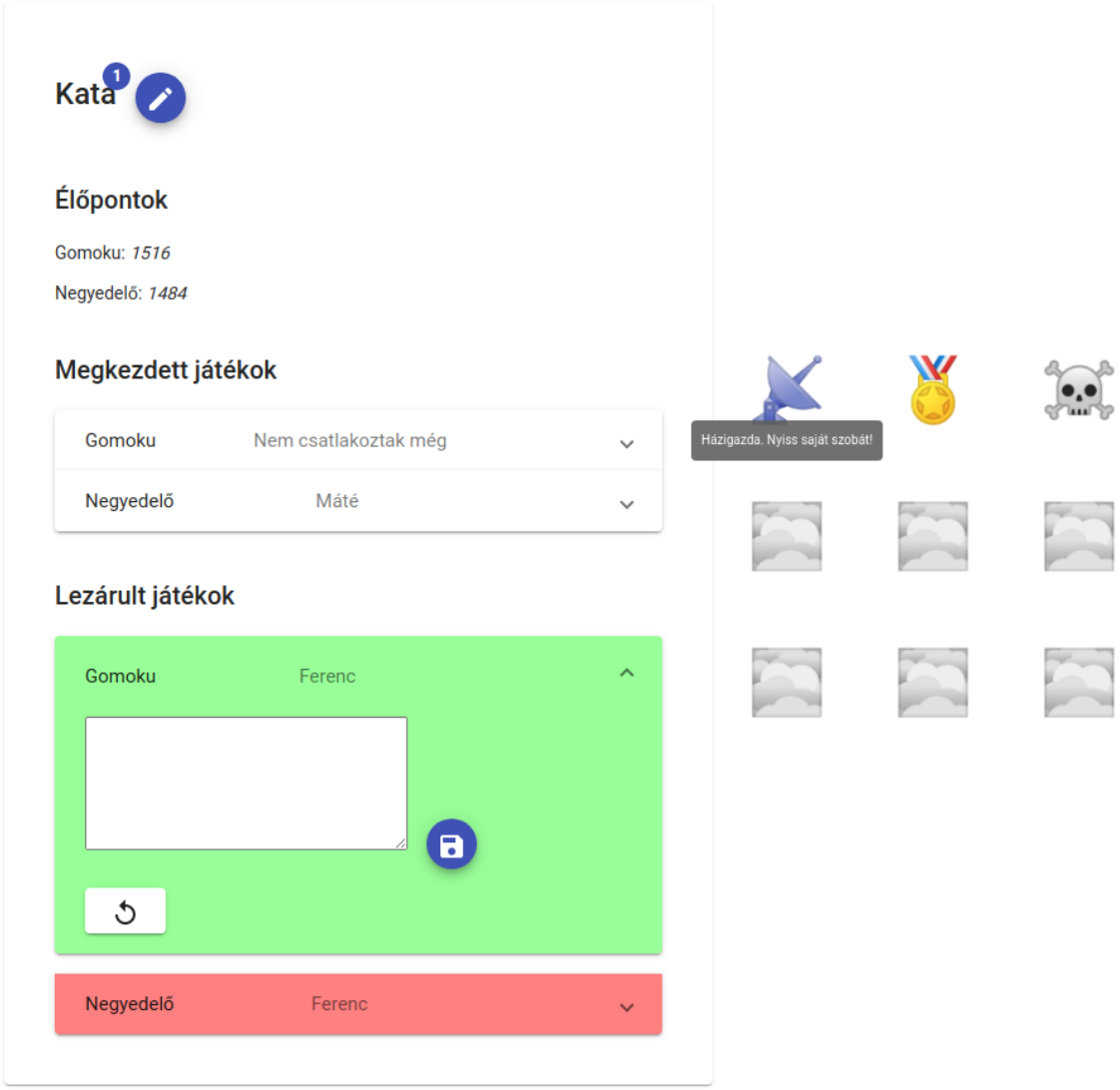
A profil oldal a felhasználó személyes lapja, ahol a neve melletti toll ikonra kattintva tud nevet változtatni.

A név alatt a játékokra lebontott Élő-pontok találhatók. Az Élő-pontokról bővebben a 3.3.5 fejezetben írok.

Az oldalon lejjebb a már megkezdett játékok sorakoznak. Ha a szobához már csatlakozott ellenfél, a legördülő lista fejlécének közepén az ő neve olvasható. Más esetben a „Nem csatlakoztak még” felirat látható. Az elemet lenyitva megjegyzést fűzhetünk a szobához, vagy módosíthatjuk azt. Alul a belépés gombra kattintva térhetünk vissza a partihoz.

Legalul szerepelnek a lezárult játékok. A piros sorok jelzik az elvesztett meccseket, a zölddel jelzettek pedig a győzteseket. Legördítve módosíthatjuk a szobához tartozó megjegyzést, illetve lehetőségünk van itt is visszalépni a szobába – ám ekkor játszani már nem tudunk.

A lap jobb oldalán helyezkednek el az achivementek, más néven a teljesítmények. Ezek kezdetben el vannak rejtve, egy kis köd ikon takarja őket. A feloldásukhoz bizonyos feladatokat kell megoldani. Ilyen lehet például, hogy nyissunk egy saját szobát, vagy hogy nyerjünk három negyedelőt egymás után. Az egeret a szimbólumokra húzva tipp jelenik meg a feladat teljesítésével kapcsolatban.

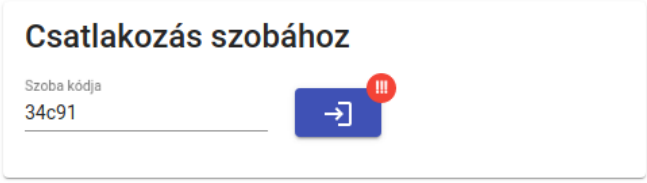


2.3. ábra. Profil oldal

2.5.5. Játékok oldal

Ha regisztrált felhasználóként ugrunk a Játékok oldalra, akkor három komponenshez férhetünk hozzá.

Az első komponens a lap tetején található. Ez a szobába való belépést biztosító kód (a továbbiakban: szobakód) beviteli mezője. Ha egy másik klienstől kapott kódot írunk be ide, akkor az átnavigál minket a megfelelő szobába. Ha helytelen kód kerül a mezőbe, akkor azt felkiáltójelek jelzik a gomb sarkában.

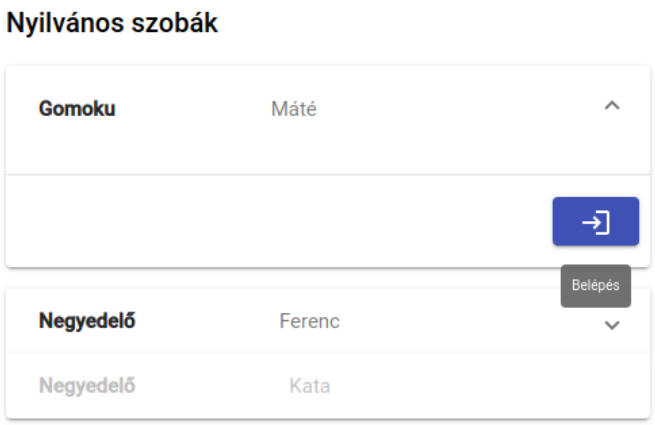


2.4. ábra. Nem létező szoba hibajelzése

A második komponens a játékok felsorolása – ezt a lap közepén látjuk. Jelenleg két játék van implementálva a **Sánc**ra. A legördülő fül fejlécében láthatjuk a játék alternatív elnevezését. Ha leengedjük a mezőt, akkor elolvashatjuk a játékszabályokat, megtekinthetjük a ranglistát és létrehozhatunk saját szobát.

Ha ez utóbbi funkcióval élünk, akkor egy felugró menüben választhatunk, hogy nyilvános vagy privát játékot szeretnénk. A nyilvános szoba megjelenik majd minden regisztrált kliensnek, a privát szobába viszont csak az kap hozzáférést, akihez eljuttatjuk a szoba kódját.

Az oldal alján található meg a harmadik komponens, vagyis azok a minden kliens számára elérhető szobák, amelyeket nyilvánosra állítottak a szobákat létrehozó felhasználók. A saját nyilvános szobáink is megjelennek a listán, azonban erről a felületről nem lehet belépni. Ide bejutni a profil oldal megkezdett játékok részénél lehetséges.



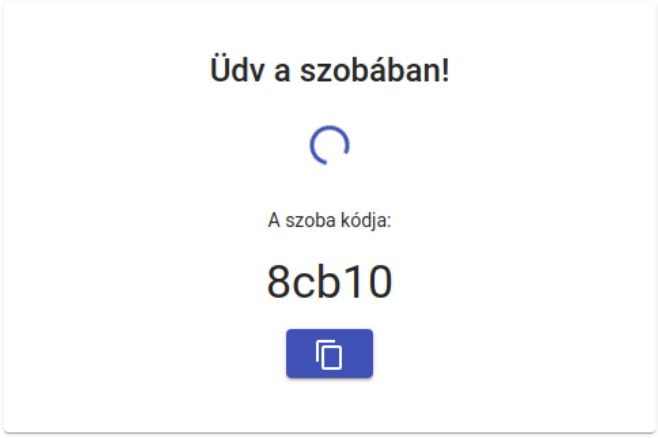
2.5. ábra. Nyilvános szobák

A Játék fül nem regisztrált (vendég) felhasználók számára is elérhető. Viszont a vendégek számára nem jelenik meg a szobakód input mező, és nem elérhetőek a nyilvános szobák. A ranglistát azonban ők is látják. A belépés gomb helyén a regisztrációra és a bejelentkezésre mutató linkek kerülnek.

2.5.6. Szobák

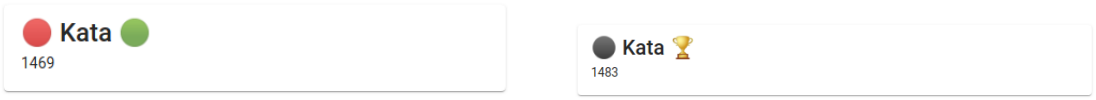
Akár gomokuval, akár negyedelővel játszunk, a szobáknak vannak közös alkotóelemei.

Amíg valaki be nem lép a szobába, addig egy töltőképernyő látható a szobakóddal. A szoba kódja automatikusan a vágólapra másolható, ha az alatta lévő gombra nyomunk. Ugyanez a felület jelenik meg akkor is, amikor egy olyan szobába szeretnénk visszalépni, amelyben a játék már megkezdődött.



2.6. ábra. Várakozás másik játékosra

A játéktábla helyezkedik el középen, felette és alatta található egy-egy játékos panel. Az alul lévő játékos panel mindig a bejelentkezett felhasználót mutatja, a felül lévő az ellenfelet. A játékkos panelen szerepel a játékos neve és Élő-pontja. A név melleti ikon jelzi, hogy melyik színnel van. Az, hogy a játékos sorra került, egy zöld ikon jelzi.



(a) Kata pirossal van, ő jön (b) Kata nyert

2.7. ábra. Játékos komponens

A játék befejezésével trófea ikon jelzi a győztest. Az oldal alján megjelenik az ereményhirdetés, illetve egy visszajátszás panel. Ezen navigálva tudjuk újra megtekinteni a mérkőzést.

2.5.7. Gomoku

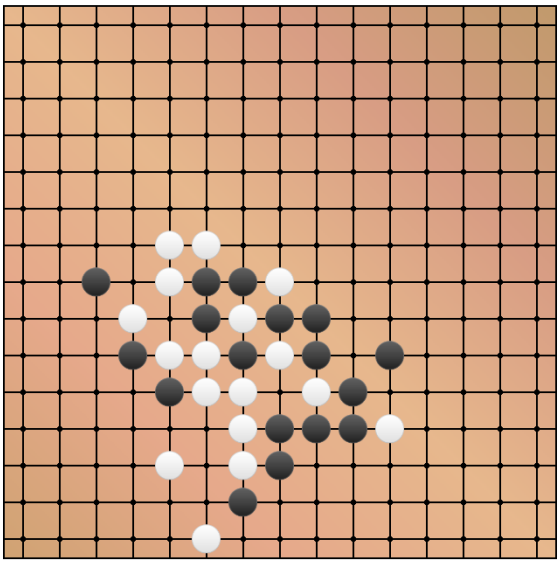
Játékszabályok

A gomokuban a játékosok fekete és fehér köveket helyeznek egy rácsozott tábla rácspontjaira. Felváltva lépnek, az nyer, akinek először sikerül egy függőlegesen, vízszintesen vagy átlóban összegyűjteni ötöt a saját köveiből.

Komponens

A gomoku tábla bármelyik mezőjére húzzuk a kurzorunkat, a rácspont sötét árnyalattal jelenik meg. Ez azért fontos, hogy minden esetben egyértelmű legyen, melyik rácspontra fogjuk helyezni a kövünket.

Kattintásra a kő lehelyezésre kerül, az aktív játékost jelölő idon pedig a másik játékoshoz kerül. Szabálytalan lépés esetén – ilyen például ha nem mi vagyunk lépésen, vagy egy már lehelyezett kőre kattintunk – egy felugró üzenet jelenik meg.



2.8. ábra. Gomoku

Ha nyerő lépést tesz valamelyik játékos, megjelenik a trófea ikon, az eredményhirdetés és a visszajátszás panel. Amennyiben ezek után kattint valaki a táblára, felugró üzenet jelzi, hogy a szoba nem fogad több lépést.

2.5.8. Negyedelő

Játékszabályok

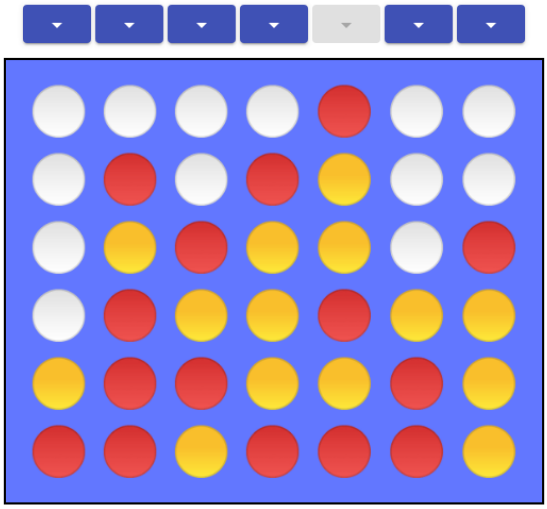
A negyedelőben a játékosok piros és sárga korongokat dobnak egy felállított táblába. Egy korong addig hullik lefelé, amíg egy alatta elhelyezkedő korongig, vagy a tábla aljáig nem ér. A játékosok felváltva lépnek. Az győz, akinek előbb sikerül függőlegesen, vízszintesen vagy átlósan összegyűjteni négy saját korongot.

Komponens

A negyedelő tábla tetején lévő gombokra kattintva tudunk bedobni egy korongot a táblába. Ha egy oszlop megtelt, az ehhez tartozó gomb inaktívvá válik.

Nyerő lépés esetén a gomokuhoz hasonlóan aktiválódnak a játék végi panelek.

A szabálytalan lépés elsősorban azt jelenti, hogy nem a sorra kerülő játékkos kattint rá a gombra. Ilyenkor egy hibaüzenet ugrik fel. Akkor is hibaüzenet érkezik, ha egy szoba már bezárt.



2.9. ábra. Negyedelő

2.5.9. Admin oldal

Ha regisztráció nélkül, vendégként nyitjuk meg a `/admin` url címet, akkor a weblap automatikusan átirányít minket a bejelentkező felületre. Ha ugyanezt regisztráltként – de nem adminként – tesszük, akkor a következő felirat fogad: „Ez az oldal csak adminoknak érhető el.”

Az admin fiókba a következő emailcím–jelszó párossal tudunk belépni:

`admin@email.hu`

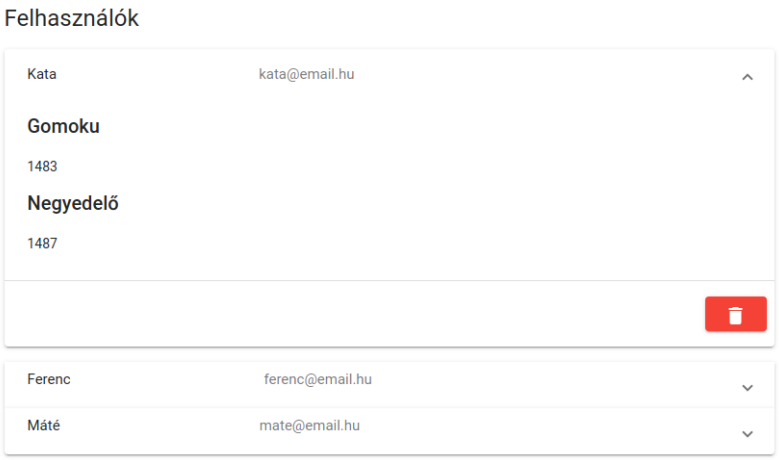
`jelszo`

Az admin valamennyi, a regisztrált felhasználó által birtokolt jogosultsággal rendelkezik. Ha az admin a profil oldalára navigál, a neve mellett egy piros ikon látható. Ez mutat az admin oldalra.

2.10. ábra. Admin oldalra vezető gomb

A betöltött lapon hozzáférünk a felhasználók és a szobák adataihoz.

Az egy felhasználóhoz tartozó legördülő panelt megnyitva hozzáférhetünk az ő különböző játékokhoz tartozó Élő-pontjaihoz. Ugyanitt van lehetőségünk a törölni a felhasználót. Amennyiben töröljük, az összes hozzá tartozó szoba is törlődik.

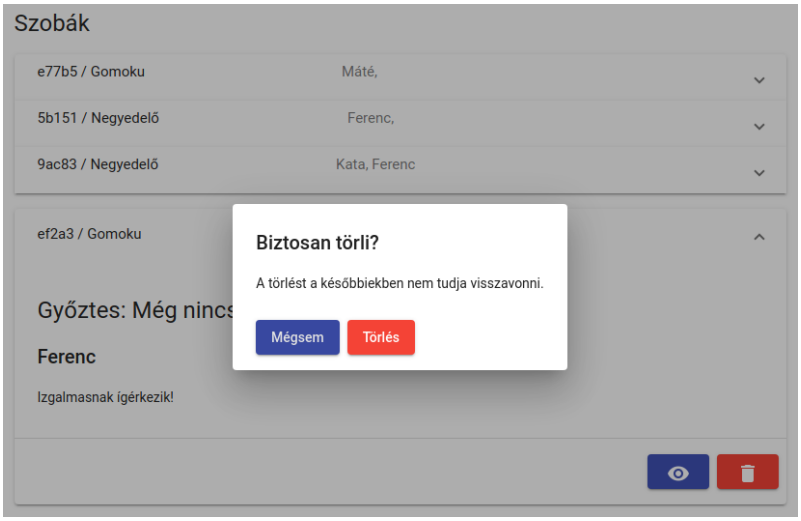


2.11. ábra. Felhasználók

Ezen az oldalon lejjebb a szobák láthatók. Minden szobának a fejlécben látszik a kódja, a játéka, a játékosai. Az elemet lenyitva megtudjuk, ki a győztes, valamint elolvashatjuk, milyen megjegyzéseket fűztek a játékosok a szobához.

A legördülő felület alján két gomb található. Ha a szobában a játék még nem kezdődött el, a szem ikon át van húzva és inaktív. Ha a játék elkezdődött, rákattintva belenézhetünk a szobában zajló játékba. Mellette törlés gomb található.

Akár felhasználót, akár szobát szeretnénk törölni, felugró ablak kérdezi meg, hogy biztosak vagyunk-e a szándékunkban.



2.12. ábra. Szoba törlése

Az admin oldal legalján egy frissítő gomb helyezkedik el, ezzel bármikor újra le tudjuk kérdezni a frissített adatokat.

2.5.10. Hibaüzenetek

Az oldal a szervertől visszakapott legtöbb hibaüzenetet lekezeli. Ilyenek például a hibásan kitöltött bejelentkező vagy regisztrációs formok, a szabálytalan lépés, a lezárt szobában történő lépés.

Ha a szerverre túl sok kérés érkezik, és nem bírja feldolgozni őket, egy várakozó komponenst ugrik fel. Ismeretlen hiba esetén azt egy felirat jelzi az oldalon.

2.13. ábra. Szerver túlterhelés

Ha egyéb, váratlan hiba keletkezne a programban, érdemes a következő lépéseket elvégezni:

1. Nyissuk meg a böngésző konzolját (F12) és olvassuk el az üzenetet.
2. Frissítsük az oldalt.
3. Próbáljunk meg kijelentkezni, majd bejelentkezni.

3. fejezet

Fejlesztői dokumentáció

3.1. Megoldási terv

3.1.1. Felépítés

Az alkalmazás három rétegen helyezkedik el.

Az első réteg a kliensoldal, a frontend. Ez elsősorban azért felel, hogy a felhasználó ergonomikusan tudjon kapcsolatot tartani a kliensoldal mögött levő rétegekkel. Itt kell megvalósítani a látványelemeket, a kényelmi funkciókat, de érdemes lehet itt elvégezni minél több számításigényes, nem szenzitív adat kezelését is.

A következő réteg a szerveroldal. Ezen a rétegen végzi el a program a nagyobb erőforrást igénylő feladatokat. Ennek a rétegnek hozzáférése van az adatbázishoz, melynek adatait le tudja kérdezni, és továbbíthatja a kliensoldal számára. Képes a lekérdezett adatokkal műveleteket elvégezni. A szerveroldalon kell a szenzitív adatokkal dolgozni. Ilyen szenzitív adat például a többi felhasználó email címe. Ha egy felhasználó szeretné megnézni a saját adatait, nem küldhetjük el az egész users táblát neki, mert ez adatkezelésileg problémás, és mert nem hatékony ilyen nagy mennyiségű adatot átküldeni, amikor csupán egy rekord is elegendő. A szerveroldal a kapcsolattartó réteg az adatbázis és a frontend között.

A leghátsó réteg az adatbázis, itt történik az adatok tárolása. Alkalmazása bonyolultabb adatbázis-kezelési feladatot nem igényel, mert a szerveroldal úgynevezett ORM-et használ, ami egy kényelmes módja az adatbázis lekérdezések elkészítéseinek. Fontos, hogy az adatbázisunkat jó struktúrában hozzuk létre, hogy az a későbbiek-

ben könnyen karbantartható és skálázható legyen.

3.1.2. Eszközök

Laravel

A *Laravel* napjaink egyik – talán a legnépszerűbb – szerveroldali keretrendszere.[1] PHP programozási nyelvet használ, ami mind a mai napig megkerülhetetlen a szerverek világában. Segítségével jól elkülöníthetők az alkalmazás routjai, modelljei, controllerjei stb. Nagy segítséget nyújt az adatbázishoz való kapcsolódásban, valamint az adatbázis és a migrációk kezelésében is. Sok kényelmi és haszonfunkciót valósít meg, ami nagyban megkönnyíti a munkát.

Angular

A Google által fejlesztett *Angular* piacvezető JavaScript keretrendszer.[2] Komponensek létrehozásával a weboldalunk apróbb, kezelhetőbb részekre bontható. A komponensek újrahasznosíthatók. Az *Angular* elkülöníti a komponens dokumentumszerkezetét (HTML), kinézetét (CSS) és funkcionalitását (JS) – és mindezt objektumelvű módon teszi, ezáltal átláthatóvá téve a kódot.

SQLite

Az adatbázishoz *SQLite* fájlalapú relációs adatbáziskezelő rendszert használtam. Ennek nagy előnye, hogy könnyen telepíthető, *Laravel*lel egyszerűen összekapcsolható, kényelmesen használható.[3]

3.2. Adatbázis

3.2.1. SQLite

Az *SQLite* egy gyors, megbízható, széles körben elterjedt fájlalapú adatbáziskezelő rendszer. C nyelven íródott, és nagy előnye, hogy platformfüggetlen és nyílt forráskódú. *SQLite* található minden Android és iPhone eszközön, használja a Skype, az iTunes és a Dropbox is.[3]

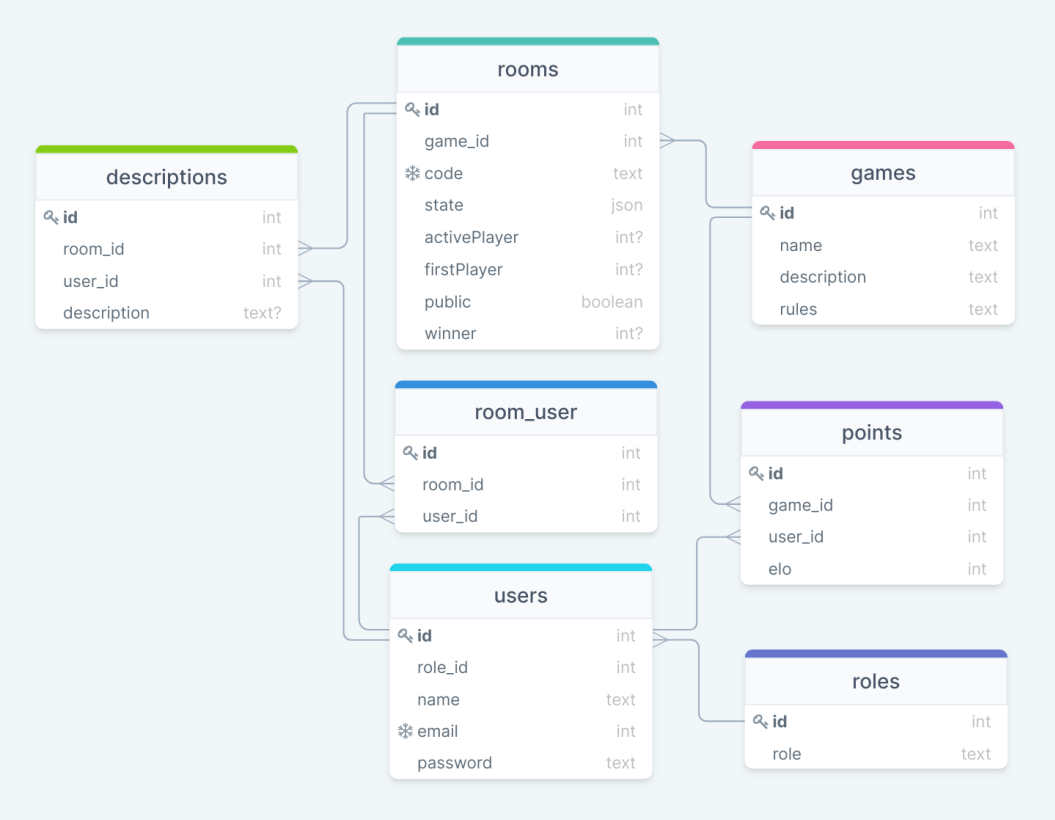
A választásom azért esett erre az adatbáziskezelő rendszerre, mert könnyen tanulható, stabil, a szerveroldalon használt *Laravel*lel hatékonyan dolgozik együtt, illetve lokális demonstrációra nem kell ennél komplexebb eszköz.

A szerveroldali projekt `.env` fájljában néhány sort átírva könnyen át lehet állítani az adatbázist más motorra, például MySql-re.[4]

3.2.2. Táblák

Az adatbázisban a két legfontosabb tábla a felhasználókat és a szobákat tartalmazza. A többi tábla az ezekhez tartozó egyéb információkat tárolja. Az adatbázisban található sok-sok és egyed-sok kapcsolatú tábla is.

3.2.3. Diagram



3.1. ábra. Adatbázis szerkezeti rajza

Az fenti diagrammon láthatjuk az adatbázistáblák grafikus reprezentációját.[2]

[2] Készült a drawSQL segítségével.

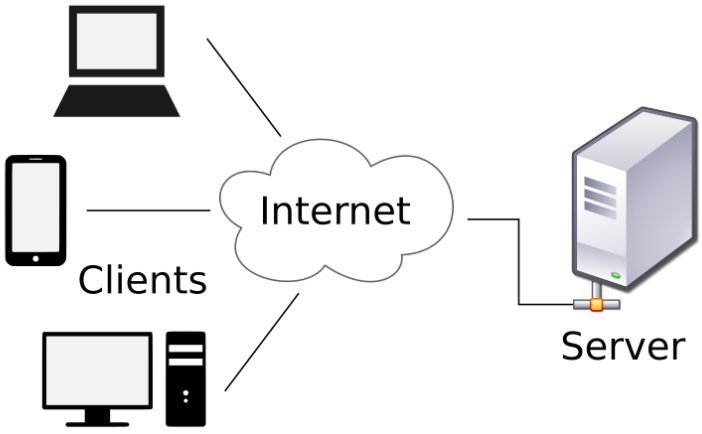
3.3. Szerveroldal

3.3.1. Laravel

A *Laravel* egy PHP keretrendszer, amellyel kényelmesen tudunk létrehozni szerveroldali webalkalmazásokat. Népszerűségét annak köszönheti, hogy gyorsan tanulható, könnyen átlátható, az interneten széleskörű közössége van.[5] Támogatja az MVC architektúrát, azaz különböző kapcsolattartó kontrollereken keresztül egyaránt támogatja a modell réteget (az adatbázist), és a view réteget (a kliensoldalt).[6, 7] A szakdolgozatomban csak modelleket és kontrollereket használok, a view szerepét a frontenden alkalmazott *Angular* keretrendszer fogja átvenni. Így az én alkalmazásom a kliens-szerver architektúrát valósítja meg, REST segítségével.

3.3.2. REST

Az alábbi ábra segít szemléltetni a kliens-szerver architektúrát:[3]



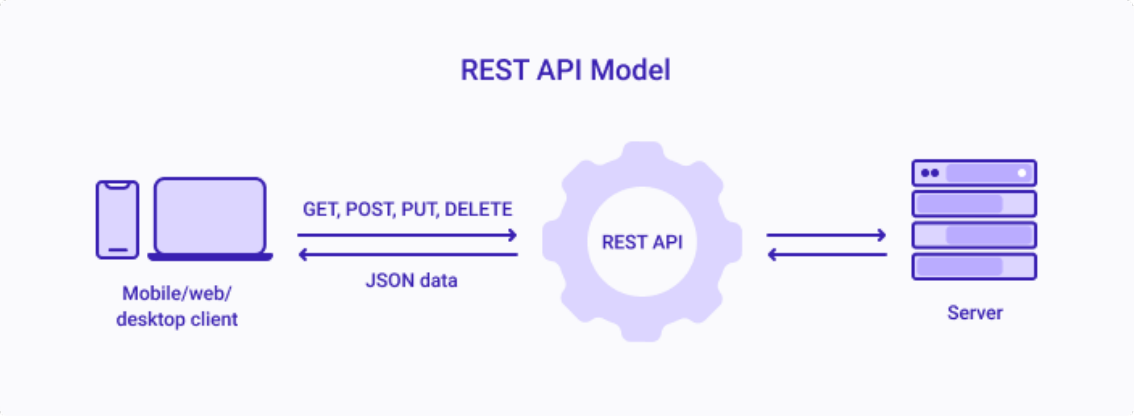
3.2. ábra. Kliens-szerver architektúra

Ahogy a rajzon is látszik, ez a szerkezet egy nagyon általános leírást ad, és csupán annyit követel meg, hogy különítsük el a szervereket a kliensektől, és ezeket az internet kösse össze valamilyen formában.

Dolgozatom esetében én egy RESTapi elkészítése mellett döntöttem, ami HTTP kérések és válaszok segítségével hozza létre a kapcsolatot.

[3] wikipedia.org [Hozzáférés: 2021. április 19.]

A REST, vagyis Representational State Transfer egy architektúra apik megvalósítására. Két fontos egységből: a kliens- és a szerverrészből áll. A szerverrel HTTP metódusok segítségével tud kommunikálni a kliens. A szerverhez gyakran tartozik valamilyen adathalmaz, amelyből a klienstől beérkezett kérésre küldhet vissza információkat. A kliens lehet egy másik program, ami ezeket a metódusokat elküldi, de a kérdések elküldését általában egy böngésző valósítja meg tetszőleges eszközön. Ahhoz, hogy egy alkalmazást RESTfulnak nevezhessünk, be kell tartania bizonyos irányelveket. Az egyik ilyen a kliens-szerver architektúra – de ilyen szabály például az szerver állapotmentessége is, amely azt jelenti, hogy nem jegyzünk meg direkt módon információt a kliensről, és minden elküldött kérés értelmezhető a korábbi kérésektől függetlenül.[8]



3.3. ábra. REST architektúra

A fenti képen látható egy REST api általános leírása.[4] A kliens valamilyen kérést küld a server irányába, ezt a REST api fogadja. A kapott kérést a szerveroldal feldolgozza, és általában egy ORM segítségével alkalmazza az adatbázisra, majd visszaküldi a feladónak megint csak a RESTapin keresztül. Egy kérés lehet XML vagy HTML formátumú is, de napjainkban a bevett megoldás a JSON, én is ezt használom a szakdolgozatomban.

Egy REST api megvalósíthatja a CRUD modellt, amennyiben a kliens részére biztosítani szeretné az adathalmaz elemein az alapvető adatkezelési műveleteket. A mozaikszó jelentése, és a hozzájuk tartozó HTTP metódusok:

[4] voximplant.com [Hozzáférés: 2021. április 19.]

- C - Create - POST
- R - Read - GET
- U - Update - PUT
- D - Delete - DELETE

Mindez annyit jelent, hogy ha a kliens írni szeretne az adatbázisba, akkor használjon POST metódust, ha pedig olvasni belőle, akkor használjon GET metódust stb. Az itt felsorolt négy művelet az adatbázis rekordok kezelésének négy alapművelete. Természetesen ezek kiegészíthetők saját műveletekkel, de a kód olvashatósága érdekében érdemes ezekhez a parancsokhoz ezeket a metódusokat alkalmazni.[9]

3.3.3. ORM

Az ORM (angolul Object-Relational Mapping) technológia lehetőséget nyújt programkódból történő adatbázis-kezelésre. Legtöbbször a keretrendszer implementálja a saját ORM-jét, de fontos, hogy az ORM se nem keretrendszer, se nem adatbáziskezelő függő, hanem egy külön átmeneti réteget alkotó eszköz a backend és az adatbázisok között. ORM-et érdemes használni, mert megkönnyíti a lekérdezések megírását, nem kell programozási nyelvet váltani – ugyanakkor ennek is meg kell tanulni a használatát, és nehezebb lekérdezések esetén lassúnak bizonyulhat.[10]

A *Laravel* keretrendszer is saját ORM-et biztosít a fejlesztők számára, ezt Eloquentnek hívják.[4] Tekintsünk meg egy példát egy egyszerű lekérdezésre Eloquent ORM segítségével:

```
1  $room = Room::where("code", $code)->first();
```

3.1. kód. Eloquent példa

A `Room` osztály a szobák modellje. Ezek közül `where` segítségével szabhatjuk meg a szűrési feltételeket. Az első paraméter az oszlop neve a szobákhoz tartozó táblában, a második – ha csak két paramétert használunk – az az érték, amivel az oszlopban szűrünk. Ha például nemegyenlőségre szeretnénk szűrni, három paramétert kellene megadunk, és ezek közül a középső lenne a `"<>"`. Ezek után a szűrés követően megkapott rekordok közül lekérjük az elsőt a `first` függvénnyel, végül értékül adjuk azt a `$room` változónak. Az Eloquent egyéb hasznos függvényeket is biztosít, például `all`

minden rekord lekérdezésére, a `find` az elsődeleges kulcs szerinti keresésre stb. Ezek a függvények tetszőlegesen kombinálhatók, ezzel kényelmes kezelőfelületet nyújtva a fejlesztő számára az adatbázishoz.[4]

3.3.4. Route-ok

*Laravel*ben könnyedén hozhatunk létre végpontokat, amelyek átirányítanak a megfelelő kontrollerek függvényeihez. Ha MVC-t alkalmaznánk, akkor a `web.php` fájlban adhatjuk meg ezeket a végpontokat, azonban szerver-kliens architektúrában az `api.php` fájl veszi át a RESTapi route kezelését alapértelmezetten.[4]

```
1  Route::group(
2    [
3      'namespace' => 'App\Http\Controllers',
4      'prefix' => 'connect_four'
5    ],
6    function ($router) {
7      Route::middleware("auth:api")->post('/move',
8      'ConnectFourController@move');
9      Route::middleware("auth:api")->get('/state/{code}',
10     'ConnectFourController@state');
11   }
12 );
```

3.2. kód. Rouetok

A példában a Route osztály group metódusát használjuk. Ezáltal van lehetőségünk végpontokat valamilyen logikus csoportosításban elhelyezni. A prefix `connect_four` azt jelenti, hogy minden megnevezett végpont előtt ez a előtag fog állni.

Middleware-nek nevezzük azokat a közbeékelt függvényeket vagy programokat, amelyek az adatfolyás közben az adattal valamilyen műveletet végeznek el.[5] Jelen példában használunk egy `"auth:api"` middleware-t. Ez egy *Laravel* által beépítetten támogatott middleware, ami azért felel, hogy csak authentikált felhasználó által küldött adat tudjon a folyamatban továbbhaladni.

[5] wikipedia.org [Hozzáférés: 2021. április 19.]

A post függvény első paramétere a végpont neve. A groupba tartozó egyik végpontunk így a:

`localhost:8000/api/connect_four/move`

, ahol a `localhost:8000` a lokális szerverünk a *Laravel* által futtatott porttal, és az `api` egy alapértelmezett *Laravel* prefix az api számára fenntartva.

Erre a route-ra a POST HTTP metódus küldése engedélyezett, ezzel más metódusokat kizártunk erről az útvonalról. Van lehetőség ugyanarra a végpontra több metódust engedni, azonban ezeket külön kell felvenni, vagy `post` metódus helyett az `any`-t használni, amivel minden HTTP kérés szabad utat kap.

A group második végpontjában kapcsos zárójelek között valamilyen argumentum várását jelezzük. Például a

`localhost:8000/api/connect_four/state/ek6hu`

végpontra küldött GET kérés az ek6hu szobának az állását szeretné lekérni.

A `post` és `get` második paramétere a végponthoz tartozó kontroller függvény meghatározása.

3.3.5. Kontrollerek

A kontrollerek alapvetően a kód rendezettségét szolgálják, vagy ahogy a *Laravel* dokumentációja fogalmaz:

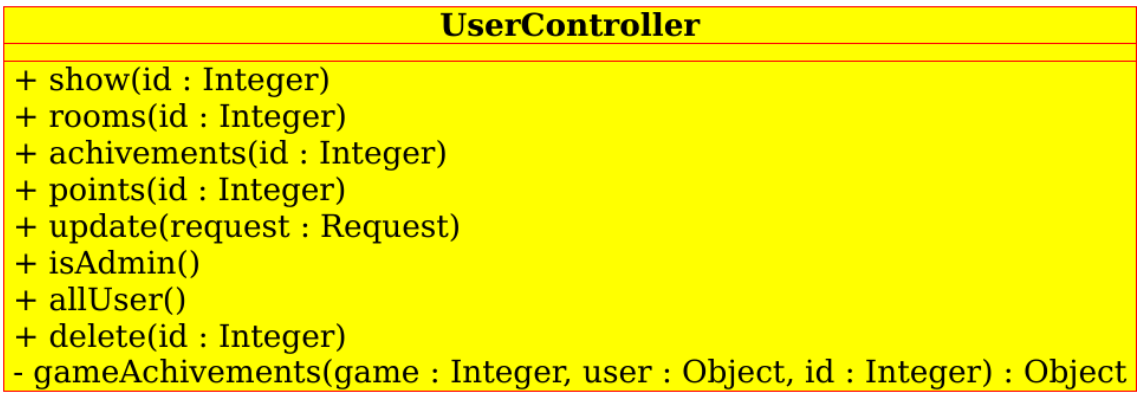
> „Instead of defining all of your request handling logic as Closures in route files, you may wish to organize this behavior using Controller classes."

Tehát a kontrollerek semmilyen különleges tulajdonsággal nem bírnak, inkább ergonómiai szempontokból érdemes őket alkalmazni.[11] Egy átlátható függvénygyűjteményt alkotnak, amelyek létrehozásakor érdemes a route-ok groupjai által meghatározott logikát követni, így az egy végponthoz tartozó függvények szervezetten lesznek a kódban megtalálhatóak.

Ha a szerver tetszőleges formátumú kérést kaphat a klienstől, akkor a kontroller feladata lehet, hogy transzformálja azt. Például ha a kérés JSON formátumban érkezik, fontos lehet a tartalmát valamilyen megfelelő adatszerkezetre parse-olni.

A szakdolgozatomban a kontrollerekben található a szerveroldal legnagyobb, legtöbb felelősséget vállaló része, ezért tekintsük át a kontrollerek fontosabb metódusait. Ezek a fájlok a `sanc/server/app/Http/Controllers` könyvtárban találhatók.

UserController



3.4. ábra. UserController

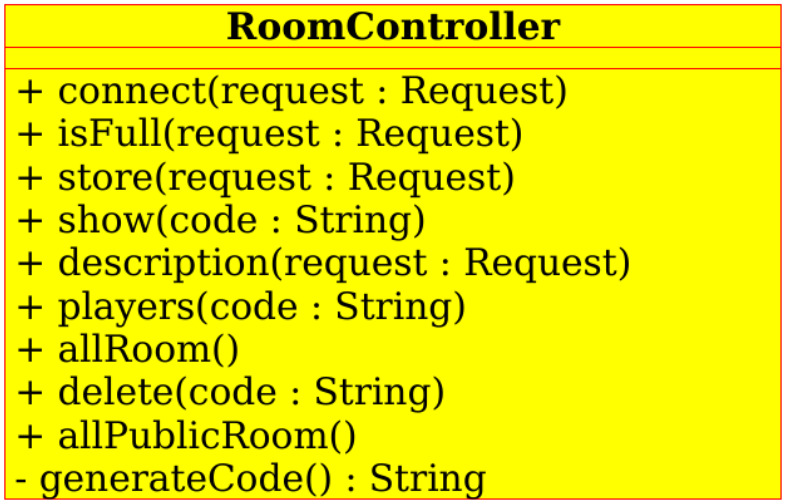
A UserController a felhasználókkal kapcsolatos információk kezeléséért felelős. Lekérdezhető az egész users tábla élőpontokkal kiegészítve, de a többi függvény mind egyedi rekordra vonatkozik. A `delete($id)` segítségével id alapján lehet felhasználót törölni, ekkor a hozzá tartozó pontok, megjegyzések és szobák is törlődnek. A felhasználónak frissíthető a neve. Az `isAdmin()` metódussal ellenőrizhető, hogy a kérést végrehajtó kliens rendelkezik-e admin jogosultságokkal. Ennek az admin oldalra való belépéskor van fontos szerepe.

Egy regisztrált felhasználó a következő achivementeket szerezheti meg a weboldalon, amennyiben teljesíti a hozzájuk tartozó feltételeket:

- Host: Nyisson egy szobát
- Győztes: Nyerjen egy játékot
- Vesztes: Veszítsen egy játékot
- Fan: Legyen legalább `x` darab befejezett az adott játékból
- Mester: Legyen legalább `y` darab nyert meccse az adott játékból
- Harcos: Legyen legalább `z` darab nyert meccse egyhuzamban az adott játékból

Az `x`, `y`, `z` háromra van beállítva, ám ezek az értékek könnyedén megváltoztathatók. A Fan, Mester és Harcos eredményekhez a privát `gameAchivements()` metódusban történnek fontos mellékszámítások.

RoomController



3.5. ábra. RoomController

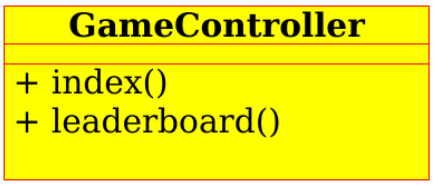
A szobához kapcsolódó endpointok kontrollere hasonló a UserControllerhez: itt is van lehetőség törlésre, összes szoba lekérdezésére (külön az összes, külön a publikus szobákéra). Itt van a csatlakozásért felelős `connect` végpont, illetve a szobába beléptetést engedélyező `isFull()` függvény. Szoba létrehozásakor a `store()` hajtódik végre. Ehhez tartozik egy `generateCode()` segédmetódus, ami a következőképpen működik:

```
1  $code = substr(sha1(time()), 0, 5);
```

3.3. kód. Kód generálás

Az adott Unix Epoch időpontot hasheljük sha1 algoritmussal.[12, 13] Ennek vesszük az első öt karakterét. Mivel a code oszlop unique, használjuk a *Laravel* Validator osztályát validálásra.[4] Addig ismételjük a kód generálását, amíg egyedi kódot nem kapunk.

GameController



3.6. ábra. GameController

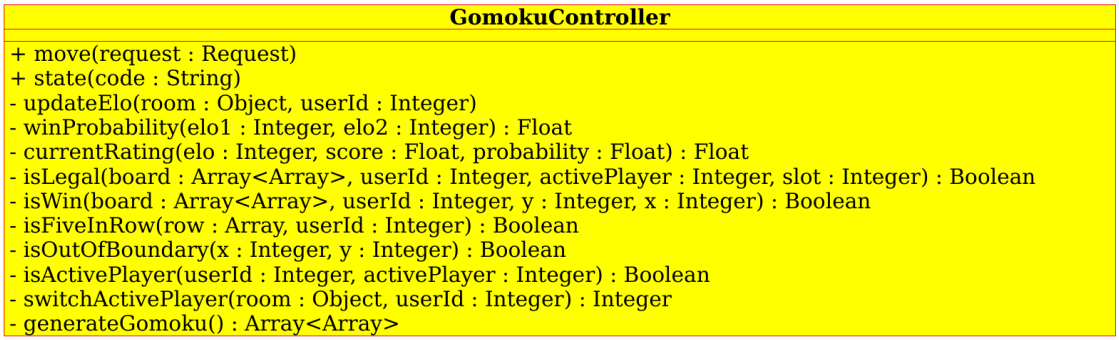
A játékok kontrollere két dologért felelős. Az `index()` metódussal lekérhető a games tábla, a `leaderboard()` metódussal pedig a játékokhoz tartozó ranglista kérhető le.

```
1  DB::table('games')->
2  join("points", "games.id", "=", "points.game_id")->
3  join("users", "points.user_id", "=", "users.id")->
4  where("game_id", $game->id)->
5      orderBy("elo", "desc")->
6      take(5)->
7      select("elo", "users.name")->get();
```

3.4. kód. Leaderboard lekérdezés

Ez az Eloquent query összekapcsolja a games, a points és a users táblát. Ezután megkeresi a megfelelő játék id-t, ezt csökkenő sorrendbe rendezi, majd lekérdezi az első öt elemét. Ebből a táblából csak az elo és a felhasználók name oszlopára van szükségünk. Így kapjuk meg az egy játékhoz tartozó ranglistát.

GomokuController



3.7. ábra. GomokuController

A GomokuController két publikus metódust valósít meg, azonban sok másik privát metódus segít a játéklogika megvalósításában. A `state($code)` visszaküldi a kliensnek a szobakódhoz tartozó játékállapotot, a `move()` függvény pedig a klienstől kapott lépést hajtja végre. Vizsgáljuk ezt meg alaposabban!

A kérés objektumnak három elemet kell tartalmaznia:

- `code` - A játék szobája
- `x` - A lépés x koordinátája
- `y` - A lépés y koordinátája

Első lépésként ellenőrizni kell, hogy a szobának van-e már győztese. Ha van, akkor 410-es Gone HTTP kódot küldünk a kliensnek, ezzel jelezve, hogy a végpont már nem fogad lépéseket. A `generateGomoku()` privát metódust segítségül hívva generálunk egy játéktábla mátrixot, amit feltöltünk az addigi játékállás lépéseivel. A játékállás eltárolása egy tömbben történik, amelynek minden eleme egy három komponensből álló objektum:

- `x` - A lépés x koordinátája
- `y` - A lépés y koordinátája
- `id` - A lépő felhasználó azonosítója

```
[
  {x: 0,  y: 0,  id:1},
  {x: 0,  y: 14, id:2},
  {x: 14, y: 0,  id:1},
  {x: 14, y: 14, id:2}
]
```

3.5. kód. Gomoku állás reprezentáció

Ha az 1. és a 2. id-jú játékosok a tábla négy sarkába tesznek köveket, azt a fent látható módon reprezentáljuk. Ennek a tárolási módnak az az előnye, hogy nem szükséges egy egész mátrixot tárolni a szerveren sok üres cellával, illetve a lépések sorrendje nyomon követhető, így a későbbiekben a visszajátszás funkció leegyszerűsödik. A kliens oldalnak is elég ezt az tömböt elküldeni, ott ebből már felépíthető a tábla

grafikus felülete, valamit az adatforgalom is kisebb, mint ha a mátrix reprezentáció mellett döntöttünk volna.

Ezek után szükséges ellenőrizni, hogy a lépés szabályos-e. Három állításnak kell teljesülnie, hogy egy lépést szabályosnak fogadjunk el. Erre szolgál az `isLegal()` metódus.

- A körön lévő játékos küldte a lépést.
- A lépés nem kerül már foglalt mezőre.
- A lépés nem kerülne a táblán kívülre.

Ha a lépés szabályos, ellenőrizzük az `isWin()` segítségével, hogy véget ért-e a játék. Ha igen, akkor frissíteni kell az adatbázisban a játékosok Élő-pontját.

Az Élő-pontrendszer egy Élő Árpád által feltalált formula, amelyet a sakk játékerő meghatározására fejlesztett ki. Azóta sok játék átvette ezt a rendszert. A **Sánc**on szereplő játékok is ezt alkalmazzák. Vizsgáljuk meg a formulát, ha A és B játékosok játszottak egymás ellen.

$$E_a = \frac{1}{1 + 10^{\frac{R_b - R_a}{400}}} \tag{3.1}$$

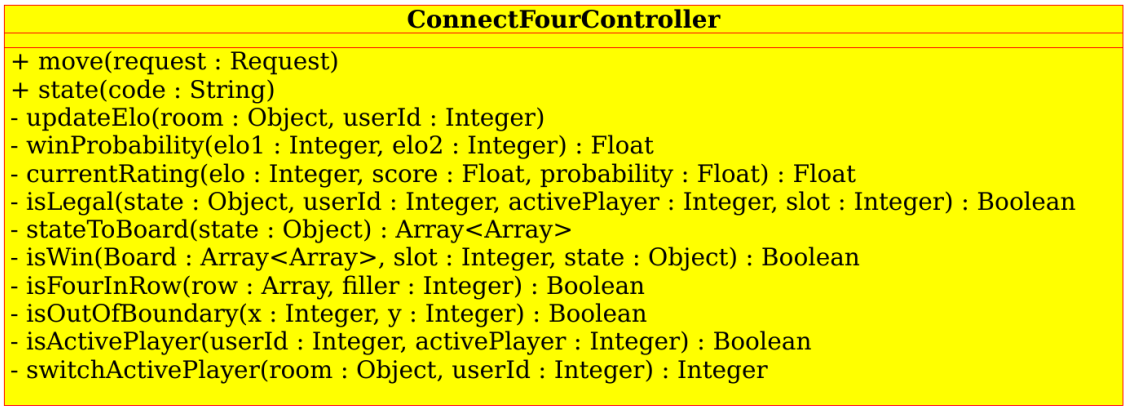
R_a és R_b A és B játékosok Élő-pontjai. E_a A játékos győzelmének valószínűsége. A 400 egy sakkban használt konstans.

$$R'_a = R_a + K(S_a - E_a) \tag{3.2}$$

, ahol S_a A játékos eredménye. Győzelem esetén $S = 1$ vesztés esetén $S = 0$. Ha lehet döntetlen eredmény is, akkor $S = 0.5$. R'_a A játékos új élője, K pedig egy önkényesen megválasztott érték, ami dinamikusan változhat bizonyos megvalósítások esetén. Ez a FIDE sakkszövetségnél függ a kortól, a játszott mérkőzésektől, az addig elért Élő-ponttól.[14] A dolgozatomban ez az érték konstans 32.[15]

Nyeréstől függetlenül, ha a lépés legális volt, az aktív játékost felcseréljük a `switchActivePlayer()`-el, és a szoba state mezőjéhez hozzátoldjuk a lépést.

ConnectFourController



3.8. ábra. ConnectFourController

A negyedelő kontrollere nagyban fedi a gomokuét. A fő különbség a játékállapot reprezentálásában található. Elég a dobásokhoz tartozó oszlopindexeket a lépések sorrendjében eltárolni.

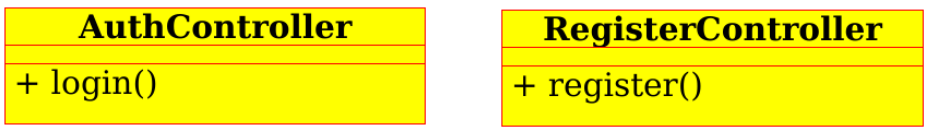
A klienstől kapott kérés a következőket tartalmazza:

- `code` - A játék szobája
- `x` - A dobáshoz tartozó oszlopindex

Legálisnak számít egy lépés, ha:

- A körön lévő játékos küldte a lépést.
- Az oszlopindex nagyobb vagy egyenlő 0 és kisebb 7.
- Az oszlop még nincs tele.

AuthController és RegisterController



(a) AuthController (b) RegisterController

3.9. ábra. Auth- és RegisterController

A RegisterController felel a regisztráló felhasználó adatainak validálásáért és elmentéséért. Az AuthController az authentikációt végzi, erről bővebben lásd 3.5

3.4. Kliensoldal

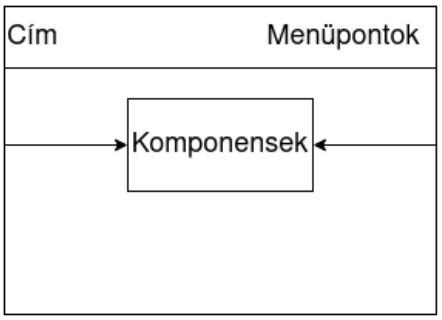
3.4.1. Angular

Az *Angular* egy Google által fejlesztett nyílt forráskódú JavaScript keretrendszer kliensoldali alakalmazások készítéséhez. Segítségével kényelmesen és könnyen fejleszthetünk komplex, komponens alapú frontend weboldalakat, tetszőleges kliens eszközre. Követi az MVC modellt, így a komponens fa jól átlátható, rendszerezett. Saját HTML template-et alkalmaz, a komponensek közötti állapotmegosztás jól kezelhető.[16, 17] A CSS alternatívájaként választható SCSS és SASS precompiler is, amely nagyban megkönnyíti a CSS formázását a weboldalon.[18] Telepítéskor hozzáférést kapunk egy command line interface-hez is (CLI), amelynek segítségével gyorsan tudunk műveleteket végezni a projektünkkel kapcsolatban, például komponenst, service-t generálhatunk, vagy elindíthatjuk a lokális szervert. Az *Angular* az egyik piacvezető JavaScript keretrendszer, az interneten sok segítség található a framework elsajátításához.

3.4.2. TypeScript

Az *Angular* alapértelmezetten TypeScriptet használ. A TypeScript a JavaScript típusos kibővítése. Minden JavaScript forráskód TypeScript forráskód, azonban ez fordítva nem igaz. A JavaScript gyengén típusos nyelv, ami bizonyos szempontból kényelmes, hiszen gyorsan alkalmazhatóvá teszi, ugyanakkor kellemetlen hibákhoz vezethet. A TypeScript segítségével lehetőség nyílik feltípusosozni a JS kódunkat (bár nem muszáj), majd compiler segítségével lefordítani azt valid JS kódra. Ha a fordító valamilyen hibát talált, akkor fordítási időben tudomást szerzünk róla, ellentétben a futtatott JavaScripttel. Másik nagy előnye, hogy a különböző JavaScript verziókra tudunk fordítani. Ezt a `tsconfig.json` fájlban tudjuk konfigurálni.[19]

3.4.3. Felhasználói felület felépítése



3.10. ábra. Drótvázterv

Az oldal mindenhol ugyanazt a drótváztervet követi, a navigációs menü minden oldalról elérhető, a konkrét tartalomhoz tartozó komponens pedig minden esetben középre van igazítva.

Az alábbi táblázat bemutatja, hogy az alkalmazásnak mely lapjai milyen jogosultságok mellett érhetők el. A nyíl szimbólum átirányítást jelent egy másik oldalra.

Útvonal	**Vendég**	**Regisztrált felhasználó**	**Admin**
`/welcome`	✓	✓	✓
`/about`	✓	✓	✓
`/login`	✓	✓	✓
`/register`	✓	✓	✓
`/games`	Korlátozott funkciók	✓	✓
`/profile`	→`/login`	✓	✓
`/room/:code`	→`/login`	✓	✓
`/admin`	→`/login`	Üzenet	✓
Egyéb	→`/welcome`	→`/welcome`	→`/welcome`

3.1. táblázat. Útvonalak elérése

Az *Angular* Single Page Application(SPA) módon valósítja meg a weboldalt. Ez azt jelenti, hogy összesen egyetlen tényleges HTML oldal van. Ha a weboldalon navigálunk, sosem új lap betöltése, csupán a megfelelő komponensek böngésző általi lerenderelése történik. Ez növeli a felhasználói élményt: nem kell minden külön oldalra lépve ismételten várni a szerverről újonnal letöltött HTML oldal megjelenésére.

Negatívum lehet viszont, hogy a webes keresőmotorok nincsenek minden esetben felkészítve az oldal tartalmában való hatékony keresésekre.[20]

3.4.4. Komponensek

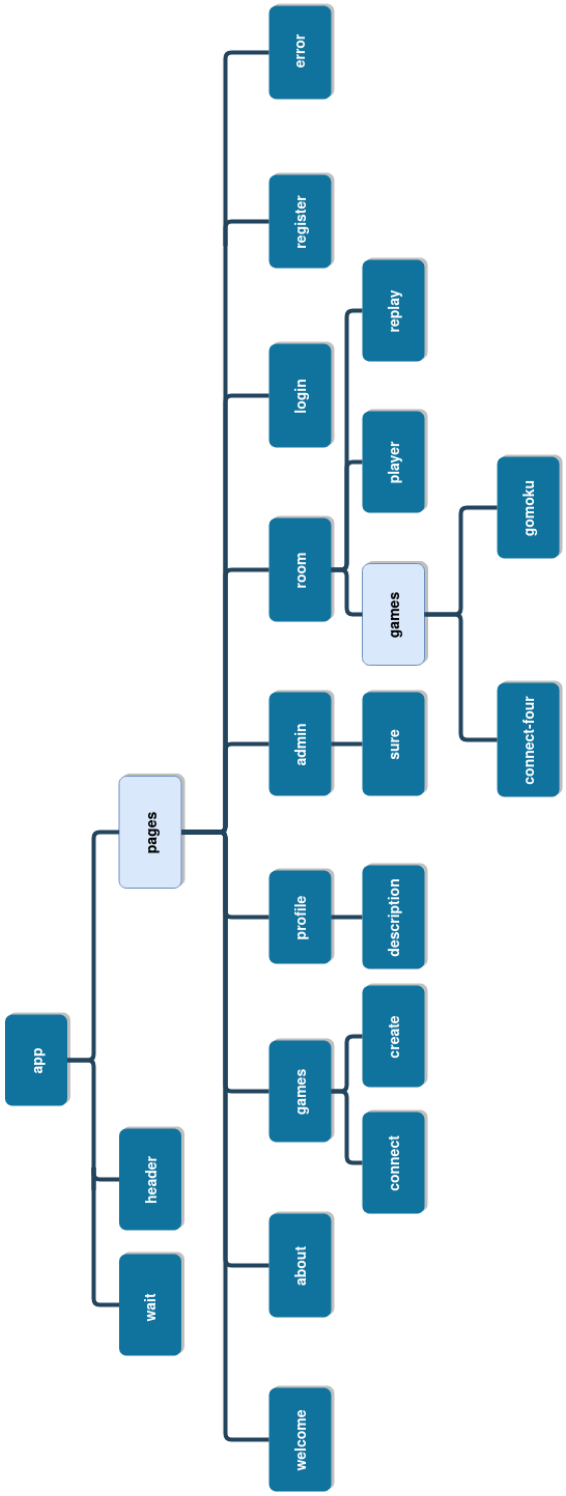
A komponensek a felhasználói felület újrahasznosítható elemei. Ha a weboldalunkon valamilyen GUI elem többször előfordul, vagy szeretnénk egy oldal elemeit modulárisan kezelni, érdemes komponenseket alkalmazni. A CLI segítségével generálhatunk saját komponenseket. Ekkor három fontos fájlt hoz létre egy paraméterként megadott nevű könyvtárban.

- `.ts` –TypeScript fájl
- `.html` – HTML template fájl
- `.css` vagy `.scss` – A formázásokat tartalmazó fájl

A TS fájl felelős a funkciók megvalósításért és az állapotkezelésért. A HTML dokumentumban template segítségével lehet hivatkozni a komponens osztály állapotát tároló adattagokra vagy függvényekre.[21] Ha ezek frissülnek, akkor az *Angular* felel azért, hogy a DOM megfelelő része reagáljon. A DOM a weblap objektumorientált reprezentációja, amit JavaScript segítségével lehet módosítani. Ha ezt tesszük, az oldal igazodik hozzá.[22]

A HTML dokumentumban a hagyományos HTML tag szintaxissal lehet más komponenseket is beemelni. A CSS logikát tartalmazó fájl alkalmazza a formázást a komponensre.

Tekintsük meg a **Sánc** komponens fáját, és néhány fontosabb komponens funkcionalitását!



3.11. ábra. Komponens fa

header

Az app komponens HTML fájljában találhatók a header és a router-outlet komponensek. Mivel a header az oldal hierarchiájában igen magasan van, ezért mindig látható, mindig elérhető. Az alatta elhelyezkedő router-outlet egy beépített *Angular* komponens, ami a hozzárendelt egyéb komponensek megjelenítéséért felelős. A **Sánc**ban ezek a pages könyvtárban találhatóak, és a különböző lapok megjelenítését végzik.

A header a navigációs menüt valósítja meg. A feladata a menüpontok a felhasználó bejelentkezésétől függő megfelelő megjelenítése, illetve a megnevezett oldalakra való átirányítás. Vannak védett oldalak, amelyek csak bejelentkezett felhasználóként érhetők el. Ezek gouarddal vannak levédve. A guard gondoskodik arról, hogy hova legyen átirányítva egy felhasználó, ha olyan oldalra szeretne lépni, amihez nincs jogosultsága.[21] A **Sánc**ban ez a bejelentkező oldal, feltételezve, hogy a kliens azért került illetéktelen címre, mert elfelejtett bejelentkezni.

login, register

A login és register komponensek HTTP POST kérés formájában küldik el a kitöltött formokat a `/login` és a `/register` szerververoldali végpontokra. A szerveroldalon az adatok validálásra kerülnek. Amennyiben a validátor errorral válaszol, a sikertelenséget üzenet jelzi a felhasználó számára.

games

Inicializáláskor a szerverről lekért adatok:

- Implementált játékok adatai
- Játékokhoz tartozó ranglisták
- Ha bejelentkezett a felhasználó, akkor a publikus szobák

A játékok aloldal megjelenítéséért felelős komponens. A felhasználó jogosultságától függően jeleníti meg a lapon részeit. Vendég felhasználó csak a játékok leírásához és a ranglistákhoz férhet hozzá, regisztrált felhasználók létrehozhatnak szobákat, vagy a már meglévőkbe beléphetnek.

profile

Inicializáláskor a szerverről lekért adatok:

- Felhasználó adatai
- Admin-e
- Felhasználóhoz tartozó szobák
- Felhasználóhoz tartozó achivementek

A profile feladata a szerverről lekért adatok statikus megjelenítése. A befejezett játékok alapján a `countLevel()` metódusban kiszámítja a játékos szintjét. Minden ötödik befejezett játék után szintlépés történik. Ha a felhasználó megszerzett valamilyen achivementet, akkor a profile komponens megjeleníti ennek grafikáját.

room

Inicializáláskor a szerverről lekért adatok:

- Szoba adatai

Szerverről pollingolt adatok:

- Tele van-e a szoba

A room komponens kezeli a betöltő képernyőt, és benne helyezkednek el a játék komponensek, a játék fajtájától függően. A polling a kliens által folyamatosan elküldött requesteket jelenti a szerver számára.[6]

Ezt a technikát használom azokhoz a problémákhoz, amikor a kliensnek reagálnia kell a szerver változására. A komponens kétmásodpercenként újraküldi az adott kérést. Ha a komponens bezáródik, a polling eseményről is leiratkozik az alkalmazás.

[6] wikipedia.org [Hozzáférés: 2021. április 19.]

gomoku

Room szülő komponenstől örökölt adatok:

- Játékosok
- Játékállás
- Szobakód

Szerverről pollingolt adatok:

- Játékállás
- Aktív játékos
- Győztes

A gomoku komponens feladata a gomoku játék lebonyolítása két játékos között. A `/gomoku/move` végpontra küldi a lépést, és a `/gomoku/state` végpontról kérdezi le kétmásodpercenként a játék állapotát az adatbázisból. Ha változik a játékállapot, a komponens adattagjai megváltoznak.

connect-four

Room szülő komponenstől örökölt adatok:

- Játékosok
- Játékállás
- Szobakód

Szerverről pollingolt adatok:

- Játékállás
- Aktív játékos
- Kezdő játékos
- Győztes

A gomoku komponenshez hasonlóan működik, a `/connect_four/move` végpontont keresztül tud lépést végrehajtani, a `/connect_four/state` végponton keresztül pedig a játék

replay

Bármely játék szülő komponenstől örökölt adatok:

- matchLength

Ez a komponens minden játékban újra felhasználható. A szülője minden esetben valamelyik játék komponens. Inicializáláskor a végetért játék hosszát (matchLength) kapja meg a replay egy saját adattagja, a replayPosition. E változó értéke a nyilak megnyomása hatására csökken, nő, nullázódik vagy áll vissza a matchLength hosszára. Változás után minden esetben küld egy eseményt a szülőkomponens számára, amelyre az fel van iratkozva. Ha ez az esemény megtörténik, a szülő újrarendereli a játéktáblát, de csak a replayPosition-edik lépésig. Amennyiben kiindexelnénk a befejezett játék hosszából, az esemény nem kerülhet elküldésre, illetve a megfelelő nyilakhoz tartozó gombok inaktiválódnak.

player

Room szülő komponenstől örökölt adatok:

- Játékosok
- Játékos szimbóluma
- Aktív játékos
- Győztes

A player komponens jeleníti meg a játékot játszók adatait a játék komponens fölött és alatt. A room komponenstől örökölt adatok alapján jeleníti meg a szükséges információkat a játékosokról. Függ a szülőjének az állapotától – ha például ott megváltozik az aktív játékos, akkor a players panelen automatikusan frissül az aktív játékos jelölő.

3.4.5. Látvány

Material

A Material egy a Google által kifejlesztett design kézikönyv, amely főleg web- és mobilalkalmazások számára készült. Fő célkitűzése, hogy letisztult, minimalista,

könnyen átlátható felületet biztosítson a felhasználóknak. Dokumentációjában — többek között — részletesen taglalja a színharmóniákat, a tipográfiát, a hangokat és az animációkat.[23] Több megvalósítása is létezik, a szakdolgozatomban az Angular Material-t használom, ami az *Angular* projektekkel kompatibilis.[7]

A különböző Material design elemekhez *Angular*ban megvalósított komponensek tartoznak, így alkalmazásuk nem bonyolult, csupán be kell importálni őket, majd HTML tag szerűen (ahogy az általunk létrehozott többi komponenst is) alkalmazhatjuk.[24] Következetes használata egységes látványt biztosít az alkalmazásoknak. Ikonkészlet is jár vele, ezek könnyen azonosíthatók a felhasználók számára.[25]

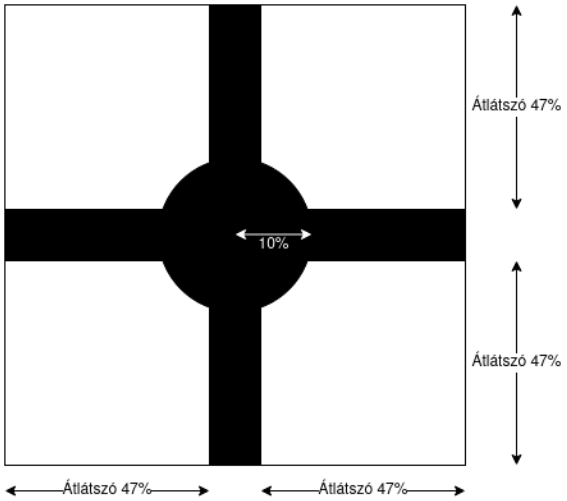
Emojik

Dolgozatomban több helyen is emojikat alkalmazok statikus grafikák helyett. Napjainkban az emojik minden nagyobb platformon támogatva vannak, így használatuk egyszerű, karakterekként lehet őket alkalmazni. Negatívum, hogy különböző implementációkban más grafika társul hozzájuk, így nem lesz egységes az alkalmazás megjelenése különböző platformokon. Az emojik alkalmazása mellett azért döntöttem, mert így nincs szükség a grafikák eltárolására, és mivel legtöbbször csak egyszerű funkciókat látnak el (fekete kő, piros korong, stb.).

Gomoku tábla

A gomoku játék megjeleítése HTML táblán történik. A tábla négyzetrácsos megjelenése jó lenne az iskolából ismert O – X megvalósításhoz, azonban a hagyományos gomokuban rácspontokra helyezik a köveket. A rácspontok létrehozásához CSS színátmeneteket alkalmazok. Azért preferálom ezt a megoldást, mert ez a CSS kód csak a hátteret módosítja, és nem rendezi át egy cella tartalmát.

[7] www.wikipedia.org [Hozzáférés: 2021. április 19.]

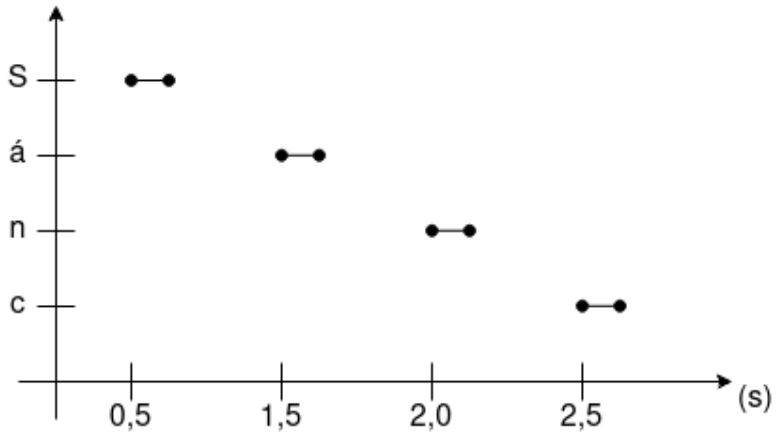


3.12. ábra. Gomoku tábla mezője

A tábla háttereként a halványbarna öt különböző árnyalata van beállítva átlós színátmenetben – ezzel fa hatást keltve a játéktáblának.

Karakter animáció

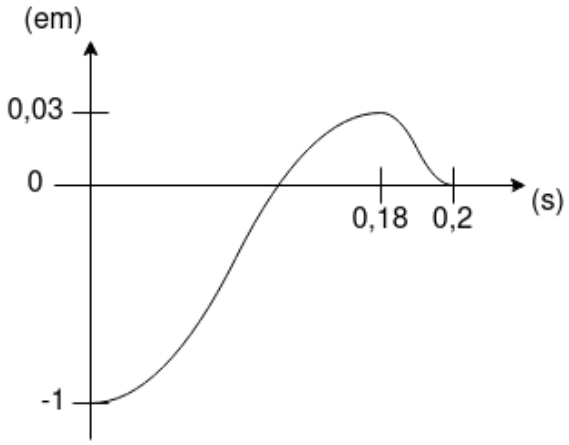
A nyitóoldalon felugró betűk CSS animációt használva jöttek létre. Az ábra a karakterek animációba történő be- és kilépését mutatja be. Az animáció az oldal megnyitásakor indul.



3.13. ábra. Karakterek belépése az animációba

A következő ábrán egy karakter mozgása látható az idő és a távolság függvényében. A távolság mértékegysége egysége em. Az em egy relatív CSS mértékegység, 1 em az elemre vonatkozó betű magasságának felel meg.[8]

[8] www.w3schools.com [Hozzáférés: 2021. április 19.]



3.14. ábra. Karakterek animációja

3.5. Authentikáció

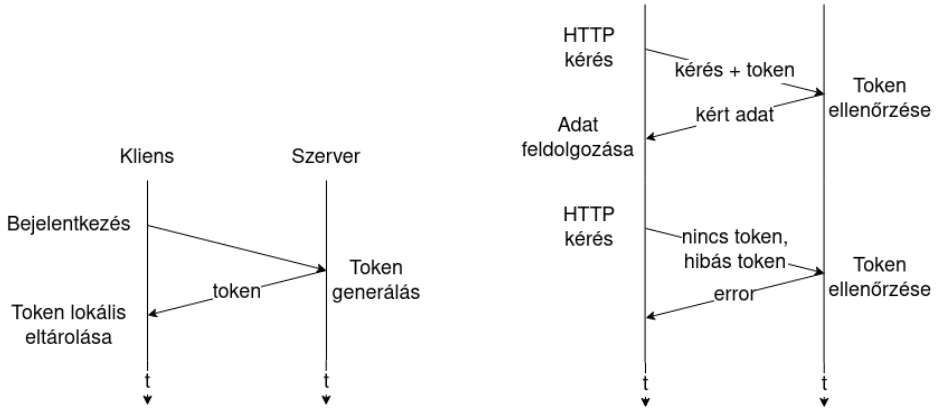
Ha egy felhasználó bejelentkezett az oldalra, akkor fontos, hogy az eszközét más eszközöktől megkülönböztessük. Ezt a folyamatot nevezzük authentikációnak. A megvalósítás JSON Web Tokennel (JWT) történik. Egy JWT három részből áll, ezek a részek pedig ponttal vannak elválasztva.

`xx.yy.zz`

Az első rész (xx) a header. Ez tárolja, hogy milyen hashelő algoritmussal van átalakítva a token tartalma, illetve milyen típusú tokent használunk. A második (yy) a payload. Ebben tárolhatunk nem szenzitív adatokat a felhasználóról, hogy azt ne kelljen minden alkalommal a szerverről lekérdezni. Itt található a token létrehozásának és lejáratának a dátuma is, ha van ilyen beállítva. A **Sánc**on alapértelmezetten egy órán keresztül él a token, utána automatikusan kilépteti a felhasználót. Fontos, hogy a payload tartalma kiolvasható a tokenből, nincs titkosítva, ellentétben a harmadik résszel (zz), a sigenature-el. Ez azért felel, hogy a szerver hitelesíteni tudja a tokent. Amennyiben a token nem megfelelő felhasználótól származik, vagy megváltoztatásra került az adatforgalom valamely pontján, a sigenature megváltozik, így a token nem lesz hitelesítve.[26]

Bejelentkezéskor a szerver a saját kulcsa szerint létrehozza a kliens számára a tokent, amit továbbít a kliensnek. A JWT állapotmentes, azaz, nincs eltárolva a szerveroldalon a felhasználó tokenje. Elég lementeni a böngészőjéhez tartozó lokális

tárhelyre a JWT-t, és minden szerverkéréshez csatolni azt. A szerver mindig ellenőrzi, hogy hiteles helyről jött-e a kérés – ha igen, teljesíti azt.



(a) Token létrehozása (b) Token ellenőrzése

3.15. ábra. Authentikáció

Szerveroldalon a *Laravel*hez készült jwt-auth package-et használom.[27] Kliens oldalon egy interceptor[21] segítségével csatolom minden kérés mellé a böngésző lokális tárolójában tartott tokent.

3.6. Tesztelés

A tesztelést kézzel végeztem, a szerveroldalt külön, feketedobozos módon, a kliensoldalt a szerveroldallal együtt, fehérdobozos módon. A weboldalt ismerősökkel is kipróbáltam: volt, akinek megmutattam a funkciókat, volt akinek nem, és volt olyan is, hogy a kipróbálást csak kívülállóként figyeltem. Ezek után megbeszéltem az ismerőseimmel a tapasztalataikat – egy részüket továbbfejlesztési lehetőségként megjegyeztem, más részüket fel is használtam. Hibákat is fedeztünk fel így, ezeket kijavítottam.

3.6.1. Szerveroldal tesztelés

`/login`

- ✓ Érvényes adatok → 200 OK, token
- ✓ Érvénytelen adatok → 401 Unauthorized

`/register`

- ✓ Érvényes adatok → 200 OK
- ✓ Email nem egyedi → 400 Validation error
- ✓ Név, email, vagy jelszó nincs → 400 Validation error

`/user/{id}`

- ✓ Nem authentikált kérés → Error
- ✓ Nem létező user id → 404 Not found
- ✓ Létező user id → 200 OK, felhasználó adatok

`/user/rooms/{id}`

- ✓ Nem authentikált kérés → Error
- ✓ Nem létező user id → 404 Not found
- ✓ Létező user id → 200 OK, felhasználóhoz tartozó szobák

`/user/achivements/{id}`

- ✓ Nem authentikált kérés → Error
- ✓ Nem létező user id → 404 Not found
- ✓ Létező user id → 200 OK, felhasználóhoz tartozó achivementek listája

`/user/points/{id}`

- ✓ Nem authentikált kérés → Error
- ✓ Nem létező user id → 404 Not found
- ✓ Létező user id → 200 OK, felhasználóhoz tartozó élőpontok

`/user/update`

- ✓ Nem authentikált kérés → Error
- ✓ Nem létező user id → 404 Not Found
- ✓ Létező user id → 200 OK, név megváltozott

`/user/admin/isadmin`

- ✓ Nem authentikált kérés → Error
- ✓ Admin kérés → 200 OK, admin
- ✓ Nem admin kérés → 200 OK, nem admin

`/user/alluser/all`

- ✓ Nem authentikált kérés → Error
- ✓ Admin kérés → 200 OK, részletes adatok minden felhasználóról
- ✓ Nem admin kérés → 205 Method Not Allowed

`/user/delete/{id}`

- ✓ Nem authentikált kérés → Error
- ✓ Admin kérés, létező user id → 200 OK, felhasználó törölve
- ✓ Admin kérés, nem létező user id → 404 OK, Not Found
- ✓ Nem admin kérés → 205 Method Not Allowed

`/games`

- ✓ Kérés → 200 OK, játékok

`/games/leaderboard`

- ✓ Az adatbázisban több mint 5 felhasználó van → 200 OK, játékonként top 5 Élő-pontú felhasználó
- ✓ Az adatbázisban kevesebb mint 5 felhasználó van → 200 OK, játékonként a felhasználók

`/isfull/{code}`

- ✓ Nem authentikált kérés → Error
- ✓ Nem létező szoba kód → Error
- ✓ Létező szoba kód, két felhasználó csatlakozott a szobához→ 200 OK, tele
- ✓ Létező szoba kód, még nem csatlakozott a második felhasználó a szobához → 200 OK, nincs tele

`/connect`

- ✓ Nem authentikált kérés → Error
- ✓ Nem létező szoba kód → Error
- ✓ Létező szoba kód, két felhasználó csatlakozott a szobához→ 409 Conflict
- ✓ Létező szoba kód, még nem csatlakozott a második felhasználó a szobához → 200 OK, játékos szobához rendelése

`/room/store`

- ✓ Nem authentikált kérés → Error
- ✓ Kérésben nem meghatározott a szoba láthatósága → Error
- ✓ Kérésben nem meghatározott a játék → Error
- ✓ A szoba láthatósága és a játék is meghatározott → 200 OK, szoba eltárolása adatbázisban

`/room/show/{code}`

- ✓ Nem authentikált kérés → Error
- ✓ Nem létező szoba kód → 404 Not Found
- ✓ Létező szoba kód → 200 OK, szoba adatai

`/room/players/{code}`

- ✓ Nem authentikált kérés → Error
- ✓ Nem létező szobakód → Error
- ✓ Létező szoba kód, egy játékos csatlakozott → 200 OK, játékos adatai
- ✓
- ✓ Létező szoba kód, két játékos csatlakozott → 200 OK, játékosok adatai

`/room/description`

- ✓ Nem authentikált kérés → Error
- ✓ Nem létező szoba kód → Error
- ✓ Szobakód vagy felhasználó nincs a kérésben → Error
- ✓ Szobakód, felhasználó, és leírás is van a kérésben → 200 OK

`/room/allroom`

- ✓ Nem authentikált kérés → Error
- ✓ Authentikált, de nem admin kérés → 405 Method Not Allowed
- ✓ Admin kérés → 200 OK, szobák részletes adatai játékosokkal, megjegyzésekkel

`/room/allpublicroom`

- ✓ Nem authentikált kérés → Error
- ✓ Nincs publikus szoba → 200 OK, üres lista
- ✓ Van publikus szoba → 200 OK, minden publikus szoba

`/room/delete/{code}`

- ✓ Nem authentikált kérés → Error
- ✓ Authentikált, de nem admin kérés → 405 Method Not Allowed
- ✓ Admin kérés, de nem létező szoba kód → Error
- ✓
- ✓ Admin kérés, létező szoba kód → 200 OK, szoba törölve

`/gomoku/move`

- ✓ Nem authentikált kérés → Error
- ✓ Létező szobakód, végetért játék → 410 Gone

- ✓ Nem létező szobakód → Error
- ✓ Nem aktív játékos küldi a kérést → 409 Conflict
- ✓ Aktív játékos küldi a kérést, nem érvényes mezőre → 409 Conflict
- ✓ Szobakód, vagy valamelyik koordináta hiányzik → Error
- ✓ Aktív játékos küldi a kérést, érvényes mezőre, ami nem nyerő lépés → 200 OK, játékállás, nem nyert
- ✓ Aktív játékos küldi a kérést, érvényes mezőre, ami nyerő lépés → 200 OK, játékállás, nyert

`/gomoku/state`

- ✓ Nem authentikált kérés → Error
- ✓ Nem létező szoba kód → Error
- ✓ Létező szoba kód → 200 OK, játékállás

`/conncect_four/move`

- ✓ Nem authentikált kérés → Error
- ✓ Létező szoba kód, véget ért játék → 410 Gone
- ✓ Nem létező szoba kód → Error
- ✓ Nem aktív játékos küldi a kérést → 409 Conflict
- ✓ Aktív játékos küldi a kérést, nem érvényes pozícióra → 409 Conflict
- ✓ Szobakód, vagy pozíció → Error
- ✓ Aktív játékos küldi a kérést, érvényes pozícióra, ami nem nyerő lépés → 200 OK, játékállás, nem nyert
- ✓ Aktív játékos küldi a kérést, érvényes pozícióra, ami nyerő lépés → 200 OK, játékállás, nyert

`/connect_four/state`

- ✓ Nem authentikált kérés → Error
- ✓ Nem létező szobakód → Error
- ✓ Létező szobakód → 200 OK, játékállás

3.6.2. Kliensoldal tesztelése

`/welcome`

- ✓ welcome komponens megjelenik
- ✓ Sánc felirat animációja lejátszódik

`/about`

- ✓ about komponens megjelenik

`/login`

- ✓ login komponens megjelenik
- ✓ Bejelentkezés gombra kattintva kérést küld a szervernek
- ✓ Ha valid adatok voltak, átirányítás történik a profil oldalra
- ✓ Ha nem valid adatok voltak, megjelenik a hibaüzenet

`/register`

- ✓ register komponens megjelenik
- ✓ Regisztráció gombra kattintva kérést küld a szervernek
- ✓ Ha valid adatok voltak, átirányítás történik a kezdőlapra
- ✓ Ha nem valid adatok voltak, megjelenik a hibaüzenet

`/games`

- ✓ Vendég számára nem jelenik meg a szoba kód beviteli mező
- ✓ Vendég számára nem érhetőek el nyilvános szobák
- ✓ Játékok leírása és a ranglisták megjelennek
- ✓ Ha regisztrált felhasználó nem létező szoba kódot ír a kód beviteli mezőbe, megjelenik a hiba jelzés
- ✓ Ha regisztrált felhasználó létező szoba kódot ír a kód beviteli mezőbe, átirányítja oda
- ✓ Ha elérhető nyilvános szoba, az megjelenik a regisztrált felhasználóknak

`/profile`

- ✓ Vendég számára nem elérhető, átirányít a bejelentkezés oldalra
- ✓ Regisztrált felhasználó számára a szerverről lekért adatok megjelennek: játékos neve, szintje, Élő-pontjai, megkezdett és lezárult játékai, elért achivementjei
- ✓ Név megváltoztatható
- ✓ Játékhoz megjegyzés fűzhető
- ✓ Győztes lezárult játékok zölddel, vesztesek pirossal jelennek meg

- ✓ Megkezdett játékba vissza lehet lépni, átirányít a megfelelő szobába
- ✓ Befejezett játék visszajátszható, átirányít a megfelelő szobába
- ✓ Kurzort az achivementek fölé húzva megjelennek a hozzájuk tartozó leírások

`/room/:code`

- ✓ Vendég számára nem elérhető, átirányít a bejelentkezés oldalra
- ✓ Ha egy várakozó regisztrált felhasználó lép be, megjelenik a várakozó képernyő
- ✓ A várakozó képernyőn megjelenik a szoba kódja
- ✓ A szoba kódja az alatta lévő gombbal vágólapra másolható
- ✓ A játékkomponens alatt és felett megtalálhatóak a játékospanelek
- ✓ A játékospanelen megjelenik a játékos neve, melyik színnel van, mennyi az Élő-pontja
- ✓ Minden játékos a maga paneljét látja a tábla alatt
- ✓ Lépés esetén vált az aktív játékos jelölő
- ✓ Győzelem esetén trófea jelenik meg a győztes neve mellett
- ✓ Ha vége a játéknak, megjelenik az értesítő szöveg
- ✓ Ha vége a játéknak, megjelenik a visszajátszás funkció
- ✓ A visszajátszás panel gombjai csak akkor kattinthatóak, ha még lehetséges az adott irányba tekerni az állást

Gomoku

- ✓ Játéktábla megjelenik
- ✓ Játékállás megjelenik
- ✓ Játéktábla mezőjére ráhúzva a kurzort, az sötét árnyalatot kap
- ✓ Ha érvényes a lépés a játéktáblán, a kő lekerül rá
- ✓ Ha a másik játékos kövére szeretnénk tenni, felugró panel jelzi, hogy szabálytalan a lépés
- ✓ Ha nem a saját körünkben szeretnénk követ letenni, felugró panel jelzi, hogy szabálytalan a lépés
- ✓ Ha csak megtekintőként kattintunk a táblára, felugró panel jelzi, hogy szabálytalan a művelet
- ✓ Ha már véget ért a játék és úgy szeretnénk követ letenni, felugró panel jelzi, hogy a szoba már nem fogad lépéseket

Negyedelő

- ✓ Játéktábla megjelenik
- ✓ Játékállás megjelenik
- ✓ Ha érvényes a lépés, a táblán megjelenik a beejtett korong
- ✓ Ha egy oszlop már megtelt, a hozzá tartozó gomb inaktívvá válik
- ✓ Ha nem a saját körünkben szeretnénk korongot dobni, felugró panel jelzi, hogy szabálytalan a lépés
- ✓ Ha csak megtekintőként szeretnénk korongot dobni, felugró panel jelzi, hogy szabálytalan a művelet
- ✓ Ha már véget ért a játék és úgy szeretnénk korongot bedobni, felugró panel jelzi, hogy a szoba már nem fogad lépéseket

`/admin`

- ✓ Vendég felhasználó átirányítása a bejelentkező oldalra
- ✓ Regisztrált felhasználónak üzenettel jelzi, hogy csak adminok számára elérhető az oldal
- ✓ Admin hozzáfér az oldalhoz
- ✓ Megjelennek a regisztrált felhasználók adatai
- ✓ Megjelennek a szoba adatai
- ✓ Ha egy szobához tartozik megjegyzés, az olvasható
- ✓ Ha felhasználót törlünk, felugrik a megerősítés panel
- ✓ Ha szobát törlünk, felugrik a megerősítés panel
- ✓ Ha törlés történt, frissül az oldal
- ✓ Az oldal alján látható frissítés gombra kattintva frissül az oldal

Egyéb

- ✓ Átirányítás történik a kezdőlapra

4. fejezet

Összegzés

A **Sánc** applikáció egy modern keretrendszerekkel és eszközökkel megvalósított weboldal, ami lehetőséget biztosít online társasjátékok játszására. Szerveroldalon a PHP nyelven írt *Laravel* keretrdenszert alkalmazza, kliensoldalon a TypeScriptet alkalmazó *Angular*t, amelyek a maguk területein piacvezető techonlógiák. A backend és a frontend kommunikációja REST alapokon nyugszik. A felhasználói felület modern, letisztult, átlátható, ami a Material Design komponenseknek köszönhető.

4.1. Továbbfejlesztési lehetőségek

Mindenek előtt szükséges a weboldal kihelyezése nyilvánosan elérhető webszerverre, hogy tényleg távoli gépekről lehessen játszani az oldalon. Ez teszteléskor megtörtént, néhány ismerősömmel így próbáltuk ki. Az ingyenes szerverszolgáltatás néhány HTTP metódust blokkolt, lassú volt, és bizonyos CPU felhasználás felett lezárta a hozzáférést. A jövőben a stabil webszerverre való kihelyezés elengedhetetlen.

A bevezetőben említett példának szolgáló oldalak alapján, a **Sánc** továbbfejlesztésének lehetősége szerteágazó. Elsősorban a játékokhoz bevezetett időmérést lenne a legfontosabb lefejleszteni socketek segítségével, amit alkalmazva már nem lenne szükség az beérkező lépések pollingolására. Az implementálható játékok száma végtelen, és különböző játékokat különböző beállításokkal is jó lenne játszani (például táblaméret beállítása, vagy pie szabály [9] opciója azoknál a játékoknál, amelyeknél

[9] wikipedia.org [Hozzáférés: 2021. április 19.]

előnye van a kezdőnek). Egyszemélyes, vagy több mint két személyes játékokat is implementálhatóvá lehetne tenni.

Hosszútávon az emojik nem alkalmasak a grafikák helyettesítésére, érdemes lenne ezeket saját grafikákkal helyettesíteni, valamint szükség esetén meganimálni őket. A Material komponensek könnyen alkalmazhatóak, azonban érdemes lehet saját designt tervezni az oldalhoz.

A körökre osztott játékok általában telefonról is élvezetesen játszhatók, így az arra való kihelyezés és optimalizáció megfontolandó.

4.2. Köszönetnyilvánítás

Szeretném megköszönni a sok segítséget Vadász Péter témavezetőmnek. Megjegyzéseivel és iránymutatásával nagyban hozzájárult a szakdolgozatom elkészültéhez.

Irodalomjegyzék

[1] Sushil Kumar Tripathi. "7 Most Popular Backend Web Development Frameworks in 2020". (2020. júl.). URL: https://www.kelltontech.com/kellton-tech-blog/7-most-popular-backend-web-development-frameworks-2020 (elérés dátuma 2021. 05. 04.).

[2] Shanhong Liu. "Most used web frameworks among developers worldwide, as of early 2020". (2021. ápr.). URL: https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/ (elérés dátuma 2021. 05. 04.).

[3] *SQLite dokumentáció*. URL: https://www.sqlite.org (elérés dátuma 2021. 04. 19.).

[4] *Laravel dokumentáció*. URL: https://laravel.com/docs/8.x/database (elérés dátuma 2021. 04. 19.).

[5] Matthaus Gorniak. "Laravel 6 PHP Framework Tutorial — Full Course for Absolute Beginners". (2019. okt.). URL: https://medium.com/@matthausz/laravel-6-php-framework-tutorial-full-course-for-absolute-beginners-5a0fa703e471 (elérés dátuma 2021. 04. 19.).

[6] Neo Ighodaro. "How Laravel implements MVC and how to use it effectively". (2018. máj.). URL: https://blog.pusher.com/laravel-mvc-use/ (elérés dátuma 2021. 04. 19.).

[7] Matt Stauffer. *Laravel: Up & Running: A Framework for Building Modern PHP Apps*. O'Reilly, 2016. ISBN: 9781492041214.

[8] Shif Ben Avraham. "What is REST — A Simple Explanation for Beginners, Part 1: Introduction". (2017. szept.). URL: https://medium.com/extend/

what-is-rest-a-simple-explanation-for-beginners-part-1-introduction-b4a072f8740f (elérés dátuma 2021. 04. 19.).

[9] Brenden Thornton. "What's a CRUD App?": (2019. aug.). URL: https://medium.com/@thorntonbrenden/whats-a-crud-app-e5a29cce03b5 (elérés dátuma 2021. 04. 19.).

[10] Berkin Bengisu. "What is Object-Relational Mapping (ORM)?": (2020. júl.). URL: https://blog.devgenius.io/what-is-object-relational-mapping-orm-177b8ab54d85 (elérés dátuma 2021. 04. 19.).

[11] Mohammad Javed. "Laravel 5.7 — Controllers". (2018. dec.). URL: https://medium.com/@mjcoder/laravel-5-7-controllers-921a546ca6f4 (elérés dátuma 2021. 04. 19.).

[12] *PHP dokumentáció*. URL: https://www.php.net/docs.php (elérés dátuma 2021. 04. 19.).

[13] Marc Stevens. *Attacks on Hash Functionsand Applications*. Leiden University, 2012. ISBN: 9789461913173.

[14] *B.01 A FIDE értékszám szabályzata*. Magyar Sakkszövetség, 2014.

[15] Matt Mazzola. "Implementing the Elo Rating System". (2020. nov.). URL: https://mattmazzola.medium.com/implementing-the-elo-rating-system-a085f178e065 (elérés dátuma 2021. 04. 19.).

[16] Dayana Jabif. "All you need to know before starting with Angular". (2018. ápr.). URL: https://medium.com/learn-angular/all-you-need-to-know-before-starting-with-angular-a7a79d556a17 (elérés dátuma 2021. 04. 19.).

[17] Minko Gechev. *Getting Started with Angular - Second Edition*. Packt, 2017. ISBN: 9781787125278.

[18] Rachel Bautista. "CSS Precompilers: Which One is Better?": (2017. ápr.). URL: https://medium.com/@raaechelb/css-precompilers-which-one-is-better-5e6ba831c0ed (elérés dátuma 2021. 04. 19.).

[19] Uday Hiwarale. “A beginner’s guide to TypeScript (with some history of the TypeScript)”. (2020. máj.). URL: https://medium.com/jspoint/typescript-a-beginners-guide-6956fe8bcf9e (elérés dátuma 2021. 04. 19.).

[20] Shifat Jaman. “Everything you need to know about “Single-page-application””. (2019. jún.). URL: https://shifat-jaman.medium.com/single-page-application-everything-you-need-to-know-6f00d87e5130 (elérés dátuma 2021. 04. 19.).

[21] *Angular dokumentáció*. URL: https://angular.io/docs (elérés dátuma 2021. 04. 19.).

[22] *MDN Web Docs, DOM*. URL: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model (elérés dátuma 2021. 04. 19.).

[23] *Material Design dokumentáció*. URL: https://material.io/ (elérés dátuma 2021. 04. 19.).

[24] Venkata Keerti Kotaru. *Angular for Material Design: Leverage Angular Material and TypeScript to Build a Rich User Interface for Web Apps*. Apress, 2019. ISBN: 9781484254332.

[25] *Angular Material*. URL: https://material.angular.io/ (elérés dátuma 2021. 04. 19.).

[26] *JSON Web Tokens*. URL: https://jwt.io/ (elérés dátuma 2021. 04. 19.).

[27] *jwt-auth dokumentáció*. URL: https://jwt-auth.readthedocs.io/en/develop/ (elérés dátuma 2021. 04. 19.).

Ábrák jegyzéke

Kódjegyzék