

Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Automatizálási és Alkalmazott Informatikai Tanszék

Nagy Viktor

**Kártyajáték megvalósítása** ASP.NET Core és Angular platformon

Konzulens

Dr. Kővári Bence András

BUDAPEST, 2021

Tartalomjegyzék

[Összefoglaló 1](#_Toc89268521)

[Abstract 2](#_Toc89268522)

[1 Bevezetés 3](#_Toc89268523)

[1.1 Általános bevezető 3](#_Toc89268524)

[1.2 Internetes játékok, társasjátékok 3](#_Toc89268525)

[1.3 Közös munka 3](#_Toc89268526)

[1.4 Saját feladatrészek 4](#_Toc89268527)

[1.5 A dolgozat szerkezete 4](#_Toc89268528)

[2 Technológiák 5](#_Toc89268529)

[3 Tervezés, architektúra 6](#_Toc89268530)

[3.1 Specifikáció 6](#_Toc89268531)

[3.1.1 A menü rendszer 6](#_Toc89268532)

[3.1.2 A játék rendszer 7](#_Toc89268533)

[3.2 A rendszer architektúrájának felépítése 8](#_Toc89268534)

[3.2.1 Adatbázis 8](#_Toc89268535)

[3.2.2 Architektúra 9](#_Toc89268536)

[3.2.3 Use-case 9](#_Toc89268537)

[4 Program részletes bemutatása 10](#_Toc89268538)

[4.1 Backend 10](#_Toc89268539)

[4.1.1 Mikroszolgáltatások 10](#_Toc89268540)

[4.1.2 Többrétegű architektúra 10](#_Toc89268541)

[4.1.3 Adatbázis 10](#_Toc89268542)

[4.1.4 Adatelérési réteg 10](#_Toc89268543)

[4.1.5 Üzleti logikai réteg 10](#_Toc89268544)

[4.1.6 Prezentációs réteg 11](#_Toc89268545)

[4.1.7 API átjáró 12](#_Toc89268546)

[4.2 Frontend 12](#_Toc89268547)

[4.2.1 Komponsek 12](#_Toc89268548)

[4.2.2 Szervízek 12](#_Toc89268549)

[4.2.3 Pipe 12](#_Toc89268550)

[4.2.4 Guard 12](#_Toc89268551)

[4.2.5 Interceptor 12](#_Toc89268552)

[4.2.6 Snackbar 12](#_Toc89268553)

[4.2.7 WebSocket 12](#_Toc89268554)

[5 Összegzés, értékelés 13](#_Toc89268555)

[5.1 Továbbfejlesztési lehetőségek 13](#_Toc89268556)

[Irodalomjegyzék 14](#_Toc89268557)

[Függelék 15](#_Toc89268558)

Hallgatói nyilatkozat

Alulírott **Nagy Viktor**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2021. 12. 02.

...…………………………………………….

Nagy Viktor

Összefoglaló

Életünkben meghatározó szerepe van a szórakozásnak. Ennek sokféle módja lehet, azonban a 21. században az egyik legmeghatározóbb az telefonos/internetes játékok

Abstract

Ide jön a ½-1 oldalas angol nyelvű összefoglaló, amelynek szövege a Diplomaterv Portálra külön is feltöltésre kerül.

# Bevezetés

## Általános bevezető

Ma már szinte elképzelhetetlen lenne az élet informatika nélkül. Ez az, ami meghatározza az életünk rengeteg aspektusát. A mindennapjaink részévé váltak nem csak szórakozás téren, de akár megélhetési szinten is, ugyanis ma már ritka egy olyan munkahely, ahol ne használnának minimális szinten se informatikai eszközöket. Akár említhetem a szakdolgozat írását is, mert annak is konkrét követelményi vannak, hogy az adott szövegszerkesztőben mit, hogyan kell beállítani.

Ahogy halad az idő előre, úgy fejlődnek nem csak az informatikai eszközök, mint például számítógép, játékkonzolok, telefonok, hanem az informatikai szoftverek is. Erre szükség is van ugyanis mivel egyre jobban behálózza a napjainkat ezért azok a szoftverek fognak fennmaradni, amelyek a legfelhasználóbarátabbak.

Az elmúlt időszakban kifejezett figyelmet kaptak a szórakoztatás céljából készült szoftverek is. Hatalmas iparágaknak az egyik fő profilja is a játékszoftverek/konzolok fejlesztése és készítése (Microsoft, Sony, Nintendo), így megkerülhetetlen egy átlagember számára is. Egy komolyabb játék elkészítése felérhet akár egy mozifilm költségeivel, például a 2020-ban készül „Cyberpunk 2077” nevű játék fejlesztési és marketing költsége meghaladta a 300 millió dollárt is ([1]). Azonban nem csak a konzolos/számítógépes játékoknak van hatalmas népszerűsége, hanem az online böngészőn keresztül játszható játékoknak is.

## Internetes játékok, társasjátékok

Kiskoromban is rengeteg weboldal épült arra, hogy a fejlesztők által készített ügyességi/logikai és egyéb internetes játékokat összefogja. Ilyenek voltak például az „Y8” és a „Startlap” weboldalak. Ezek mind Adobe Flash technológiára épültek, amely egy az internetes grafikát megjelenítő szoftver volt. Azonban biztonsági hibái miatt ennek a szoftvernek 2021 elején megszűnt a támogatása ([2]).

Azonban itt kitérnék a társasjátékok népszerűségére is és a kapcsolatukra az internetes játékokkal. A társasjáték fogalma nagyon régre nyúlik vissza, feltárások bizonyítják, hogy már időszámításunk előtt is léteztek társasjátékok. Így nem csoda, hogy manapság is hatalmas népszerűségnek örvendenek. Eléggé gyakoriak azok a társasjátékok is melyek kaptak internetes megfelelőt is (pl.: Honfoglaló, Monopoly, Activity, Uno).

Ezért úgy gondoltuk, hogy mi is kihasználjuk ennek a két területnek a népszerűségét és megalkotjuk egy bonyolultabb társasjáték internetes megfelelőjét. A többesszám használata nem véletlen, erre a következő fejezetben ki is térek.

## Közös munka

A projektet alapvetően ketten csináltuk Markovics Gergellyel, ugyanis szerettünk volna egy minél nagyobb és részletesebb szoftvert alkotni, ami később akár ténylegesen használható lesz bárki számára. Az önálló laboratórium tárgynál már közösen dolgoztunk és könnyen tudtunk együtt haladni, így célszerűnek gondoltuk, hogy a szakdolgozathoz készült munkánkat is közösen csináljuk.

Azonban a korábbi munkánkkal ellentétben, ahol sok mindent készítettünk közösen együtt, idén úgy gondoltuk célszerűnek, hogy jobban leválasztjuk magunkat a másik munkafolyamatáról és jobban elhatárolható részeken dolgozzunk, ne függjünk attól, hogy a másik, hogy halad és mit csinál éppen. Ez az elején még nem kivitelezhető ugyanis a tervezési folyamat alatt, ahol az alapvető felépítést, specifikációt és kinézeti elemeket beszéltük meg még közösen kellett dolgoznunk.

Kérdéses volt, hogy hogyan szeretnénk szétosztani a feladatköröket. Az első ötlet az volt, hogy szétosztjuk backend-frontend vonalon azonban ezt elvetettük. Végül azt a megoldást választottuk, hogy ugyan mindketten fogunk backend és frontend feladatokat is készíteni, de inkább funkció szerint osztjuk el a munkát.

Ez azt jelentette, hogy míg a társam készítette magát a játék logikáját és kinézetét (belesegítettem a tervezés részénél és a felhasznált képek elkészítésénél), én foglalkoztam minden olyan feladatkörrel, ami nem a konkrét játékmenethez tartozik. A specifikáció rész menü rendszer pontja (3.1.1) alatt részletezem, hogy mi volt a konkrét feladat, úgyhogy itt csak felületesen térek ki rá felsorolás szerűen.

## Saját feladatrészek

Az elején közös munka részeként foglalkoztam a program alapvető tervezésén, ideértve az adatbázis kezdetleges megtervezését, a használati esetek elkészítését és a felülethez használt skiccek felrajzolását és emellett átbeszéltük, hogy milyen architektúrát lenne érdemes használnunk. Későbbi munkafolyamat során besegítettem társamnak a felületéhez tartozó kártyák képeinek elkészítésével (46 darab kártya azonos méretűre vágása lekerekített széllel, a kártyákhoz tartozó franciakártya jelölések elkészítése és egyéb kisebb képek).

A következőnek felsorolt feladatok közül a backend és frontend rész implementálása is a feladatom volt. Elsőként létre kellett hoznom a projekt alapját, tehát működnie kellett az adatbázisnak, mikroszolgáltatásoknak. A bejelentkezés és regisztráció implementálása volt a legfontosabb ugyanis ettől függött utána minden. Ezután jöhettek az egyéb feladataim, mint például a barát rendszer, a váró rendszer és egyéb ezekhez kapcsolódó feladatok. (pl. felhasználóhoz kötött eredmények mutatása, chat rendszer kialakítása)

## A dolgozat szerkezete

A bevezető részt leszámítva alapvetően 4 nagyobb egységre lehet bontani a dolgozat felépítését. Itt a bevezetőben egy általános képet adtam arról, hogy mi az alap koncepció, miről fog szólni az elkészített program.

A követekező pontban leginkább egy részletesebb követelményspecifikáció lesz, ahol beszélek a rendszer kialakításáról, az adatbázis felépítéséről, a program architektúrájáról és a használati esetekről.

A harmadik részt egy szárazabb technológiai bemutatónak gondoltam, ahol felsorolom, hogy milyen technológiákat használtam a program elkészítése során és ezek mindegyikéről részletesebben írok.

Majd ezt követi az önálló munka bemutatása. Ez egy hosszabb blokk lesz, ahol bemutatom a program szerkezetét a backendtől a frontendig, tehát a felépítése ennek a résznek nem funkciók szerinti szétosztással fog elkészülni, hanem az architektúrában lentről felfele haladva. Kitérek minden számomra érdekesebbnek vagy bonyolultabbnak talált megoldásokat.

Végül zárásként megpróbálom objektívan értékelni az elért eredményt, összegezni, hogy mire jutottunk, mik voltak a nehézségek, mit tanultam a feladat megoldása során, mire mekkora hangsúlyt fektettem. Ezen felül leírom, hogy miben lehetne még továbbfejleszteni a programot, mi az, amit terveztünk bele, de nem sikerült időben megvalósítani.

# Tervezés, architektúra

## Specifikáció

A program mivel szétosztható két részre ezért külön szedem őket, azonban a második részéről nem fogok nagyon pontos specifikációt adni, ugyanis abban csak a tervezés szintjén vettem részt, nem én implementáltam. Ugyanis ahogy azt a bevezetőben (1.3) említettem én feleltem a játék egyéb működtető elemeiért, ami nem maga a játszható játék. (pl.: authentikáció, várórendszer, menü, chat, barátrendszer stb.)

### A menü rendszer

A szoftver elindításakor egy bejelentkező felület fogad minket. Itt meg tudjuk adni a felhasználónevünket (aminek muszáj egyedinek lennie) és a jelszavunkat (kötelező tartalmaznia legalább egy nagy betűt és kisbetűt, és legalább egy számot). Azonban, ha még nem regisztráltunk nem fogunk tudni belépni, ilyenkor a „Bejelentkezés” gomb alatt át tudunk lépni a regisztrációs felületre. Itt ugyanúgy meg kell adni a felhasználónév-jelszó párost, azonban meg kell erősíteni a jelszót, ugyanis, ha a két jelszó bemenet nem egyezik nem engedélyezi a regisztrációt. Sikeres regisztráció esetén visszakerülünk a bejelentkező felületre, ahol már helyes bemenetek után be fogunk tudni jelentkezni. Innen fogunk átkerülni a menübe.

A menüt alapvetően két részre lehet osztani, a bal oldali fő részén jelennek meg azok az opciók, amelyeket végrehajthatunk, ilyen például a váró létrehozása, a váróba becsatlakozás jelszó alapján, a korábbi játékok eredményeinek megtekintése, a kijelentkezés és a profil törlése. A kijelentkezés az visszadob a bejelentkező felületre, a felhasználó törlése ugyanezt csinálja, csak mellette kitörli a felhasználót. A jobb oldalon pedig egy barátlista található.

A barátlista két felületen jelenik meg, a menüben és a lobbyban. Annyi a különbség a kettő között, hogyha a felhasználó egy varóban van akkor tud küldeni meghívást egy barátjának, hogy ő is csatlakozzon be. A lista három részre van osztva, van egy felsorolás azokról, akik ténylegesen a barátaink, van alatt egy kisebb lista rész, amely azokat az embereket tartalmazza, akik bejelöltek minket ismerősnek azonban még nem jelöltük vissza. Van legalul egy kis rész annak, hogy barátnak tudjunk jelölni új embereket név alapján (ezért is szükséges az egyedi név). Egy barátjelölést el lehet fogadni, ilyenkor bekerül a legfelső bartálistába, azonban el is lehet távolítani a jelölés kérelmet. A már bejelölt barátokat is lehetséges törölni. Ha egy váróban lévő barátunk küldött nekünk egy meghívást akkor mi azt el tudjuk fogadni, azonban, ha mi is egy váróban vagyunk erre nincs lehetőség. Ugyanez a helyzet, ha azonos váróban vagyunk, ott le is van tiltva a meghívás küldése.

Egy váróba többféleképpen is eljuthatunk, ha létrehozunk egyet magunknak, ha elfogadjunk egy barátunk meghívását vagy ha a váró saját jelszavát használva belépünk oda. Minden várónak van egy birtokosa, ő az, aki létrehozta az adott várót, az ő jogában áll elindítani a játékot. Ha elhagyja a várót akkor ez a jog véletlenszerűen át fog kerülni más, a váróban tartózkodó játékosra. A felületen láthatjuk az adott váró jelszavát, a benne tartózkodó emberek nevének listáját, egy indító gombot (csak a birtokos számára) amely csak akkor megnyomható, ha legalább 4-en tartózkodnak benne (maximum 7), és egy váró elhagyása gombot, melyet megnyomva el lehet hagyni a várót és vissza lehet kerülni a menübe.

Az eredmények egy egyszerű felület, itt egy listában fel van sorolva az, hogy az utolsó tíz játékban melyik karaktertípussal és hányadik helyezést értünk el.

A váróban és magában a játékban van egy beszélgető felület. Itt az adott váróban, majd később a játékban résztvevő játékosok tudnak egymással beszélgetni. Egy beszélgetés bejegyzés tartalmazza azt, hogy ki írta az üzenetet és magát az üzenetet.

### A játék rendszer

A játék elindítását követően megjelenik minden játékosnak maga a játéktér. A játéktér több fő elemet is tartalmaz. A saját táblánkat, a középen elhelyezkedő asztalt, ahonnan a lapokat kell húzni és visszarakni, és a többi játékos táblája.

A játék egy körökre osztott kártyajáték. Minden játékos véletlenszerűen megkap egy szerepet, azonban senki nem ismeri a másik szerepét (sheriff, renegát, bandita, sheriffhelyettes), kivétel a sheriff, mert róla mindenki tudja kicsoda. A szerep mellett mindenki kap egy karaktert, ezt már a többi játékos is láthatja. Minden karakternek különböző képességei vannak, például több életpont, több kártyát húzhat stb. A játék célja, hogy te és a veled azonos szerepeket kapó játékosok maradjanak az utolsók, akik életben maradnak. A banditák célja úgy megölni a sheriffet, hogy nem buknak le, mert ha lebuknak akkor a sheriffhelyettesek tudják, hogy kik elől kell megvédeniük a sheriffet. A renegát mindenkin kívül áll, neki csak annyi a feladata, hogy egyedül maradjon.

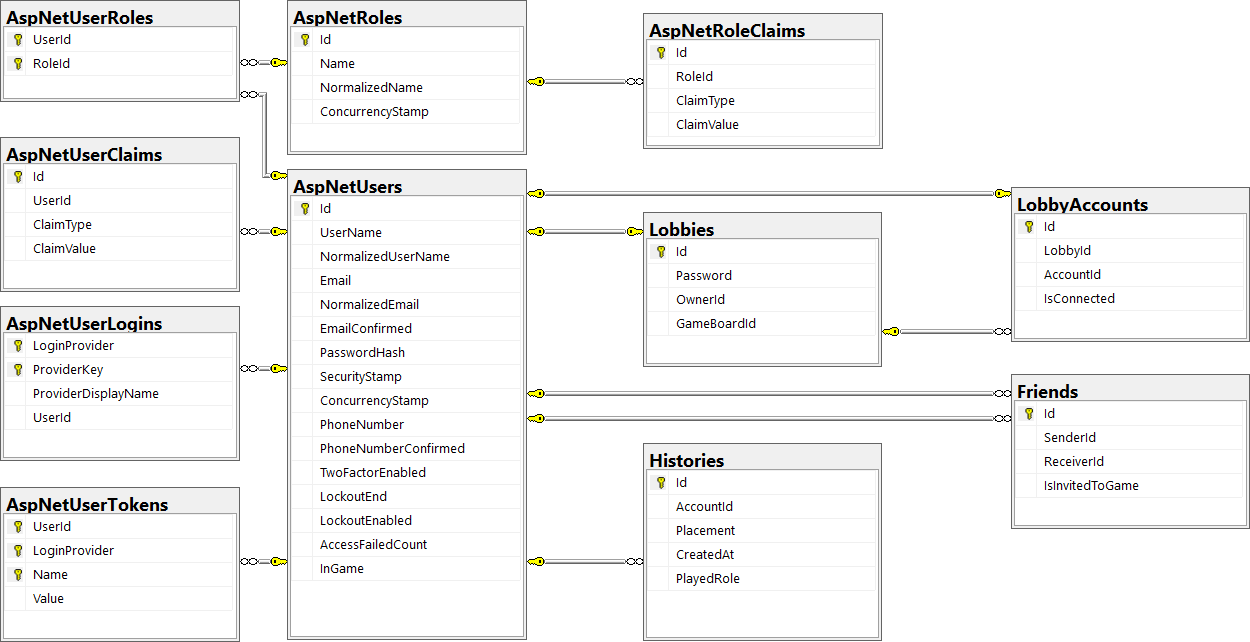
A játékot segítik elő fegyverek és egyéb lapok, amiket nem fogok részletesen bemutatni. A fegyverekkel távolabbra tudunk lőni, ugyanis a két legfőbb lap a „Bang” és a „Nem talált” ugyanis ezekkel tudjuk a többi játékos életét csökkenteni, vagy épp a saját életünket megmenteni.

Egy körben annyi kártyát játszhatsz ki amennyit szeretnél, csak arra kell figyelni, hogy maximum annyi kártya maradhat a kezedben amennyi az életerőd. Ha esetleg nem sikerült ennyi kártyát kirakni, akkor a kör vége előtt kénytelen vagy őket eldobni.

## A rendszer architektúrájának felépítése

### Adatbázis

A projekthez két darab adatbázis táblát használtunk. Ezzel is hangsúlyozva azt, hogy a társammal minél inkább próbáltuk elhatárolni egymás munkáját azonban elkerülhetetlen, hogy a két tábla között kapcsolat jöjjön létre. Elsősorban a menürendszer adatbázisáról fogok beszélni (1.ábra), ugyanyis a játék adatbázisában csak a tervezés szintjén vettem részt és a képen látható adatbázisfelépítés már a végleges verzió. Ebben a blokkban nem fogok kitérni részletesen az adatbázis kódbéli működtetésére, inkább az alapvető tervezői döntésekről ejtek szót.



. ábra: A menü adatbázisának felépítése

Az adatbázis tényleges létrehozása Code-First módszerrel készült, erre a későbbiekben fogok részletesebben kitérni. (4.1.4.2) Röviden annyi, hogy a kódban lévő osztályokat képeztük le adatbázismodellé.

A képen látszódik a sok „AspNet”-el kezdődő tábla, ezek amiatt generálódtak, mert a saját „DbContext”-ünk nem a sima „DbContext” osztályból származik le, hanem az „IdentityDbContext”-ből. Erre a felhasználók kezelése miatt van szükség azonban erről is részletesebben ki fogok térni a későbbiekben (4.1.4.3).

A „Lobbies” tábla tárolja, hogy mi az adott varó jelszava, a felhasználót aki létrehozta a várót és hozzá tartozó „GameBoardId”-t. Ez a játék adatbázisának egy külső kulcsa, ami nulla értéket vesz fel, ha nincs elindítva a játék, amint elindul a hozzá tartozó GameBoard kulcsát megkapja. A felhasználó és a váró közé kell egy kapcsolótábla, amely tárolja az adott váróban tartózkodó játékosokat, ennek a neve „LobbyAccounts”.

A „Friends” nevezetű táblának két külső kulcsa van a felhasználókra. A tábla lényegében úgy működik, hogy ha érkezik egy barát bejelölés akkor az bekerül egy küldő-fogadó rekordként a táblába, viszont csak akkor lesz elfogadva a barátság, hogyha ez a kapcsolat fordítva is létezik. Itt van eltárolva még a váróba való meghívás értéke is.

A „Histories” tárolja, hogy melyik felhasználó milyen karakterrel hanyadik helyezést ért el. Ennek az időpontja is el van mentve.

A képen szöveg, égbolt, képernyőkép látható

Automatikusan generált leírás

. ábra: A játék adatbázisának felépítése

Mint említettem a játék adatbázisának csak a tervezés szintjén vettem részt, a társam volt, aki ezt később javítgatta, hozzárakott/elvett. Alapvetően úgy épül fel, hogy létezik a „GameBoard” ami igazából a váró táblájának játékbéli megfelelője. Itt kiegészül minden olyan eseménnyel, ami előfordulhat a játékban (IsOver, TurnPhase). Vannak még a játékosok. Ez a felhasználó megfelelője a játékban, ahol kiegészül egyéb tulajdonságokkal. A játékosoknak is vannak kártyájuk, és az asztalon is van kártya, ezért ezt a feladatot két darab kapcsolótábla látja el.

### Architektúra

Két nagy részre lehet szétbontani a projektet, egy frontend részre, amely Angular technológián alapul és egy backend részre mely .NET-ben íródott. A két rész hálózati hívásokon keresztül kommunikál egymással.

A képen szöveg, computer, számítógép, sötét látható

Automatikusan generált leírás

. ábra: Architektúra felépítése

A képen látható, hogy a frontend API (application programming interface) hívásokon keresztül ráhív egy átjáróra, amely továbbítja a megfelelő mikroszolgáltatásnak a kérést. A három rétegű struktúra legfelső rétegében lévő kontroller fogadja a hívást, majd az üzleti logikai réteg megfelelő függvényére továbbítja azt. Ebben a rétegben zajlanak az adatbázishívások. Ezt segíti az adatelérési réteg, ahol definiálva vannak az entitások és az adatbázisok. Ezután az adatbázis szinttől visszaadja a megfelelő adatot a frontend számára. Annyival még kiegészíteném, hogy a Websocket technológia miatt nem csak egyirányú feltétlenül a kommunikáció, egy adott mikroszolgáltatás is tudja értesíteni a frontendet, ha valahol változás történt.

Részletesen nem térek ki az egyes egységek pontos működésére, erre a későbbiek folyamán fog sor kerülni.

### Használati esetek

Az úgynevezett „use case diagram” egy grafikus ábrázolása annak, hogy a felhasználónak milyen interakciói lehetnek a rendszerrel. Különböző használati eseteket mutat be akár több fajta szereplőn keresztül.

A képen szöveg, vezérlőelem látható

Automatikusan generált leírás

. ábra: Használati esetek

Ahogy a képen látszódik a program több oldallal/komponenssel rendelkezik. A legegyszerűbb a bejelentkező/regisztrációs felület. Innen a felhasználó létrehozása után bejelentkezve jutunk el a menü oldalra.

A menü oldalon megjelenik az 5 különböző opció, ezek közül kettő érdekes, ugyanis a „Váró létrehozása” és a „Csatlakozás a váróhoz” az a kettő mellyel tovább tudunk menni a következő oldalra a Lobby Page re.

A barátlista megjelenik a menüben és a váróban is. Ebben el tudjuk végezni az alapvető műveleteket, mint a hozzáadás, eltávolítás, meghívás elfogadása játékra. Ha az aktuális játékos egy váróban van akkor képes meghívót küldeni oda.

A váróban pedig képesek vagyunk kilépni belőle vagy üzenetet küldeni. A váró tulajdonosa pedig el tudja indítani a játékot.

A játékban pedig a szokásos elemek jelennek meg, mint például kártyahúzás, kör abbahagyása, kártya kijátszás, életerő csökkentés.

# Technológiák

## Angular

## .NET, ASP.NET

## Docker, docker compose

A Docker képes egy alkalmazást és függőségeit egy virtuális konténerbe csomagolni, amely bármilyen Linux, Windows vagy macOS számítógépen futtatható. Ez lehetővé teszi, hogy az alkalmazás különböző helyeken fusson, például helyben, nyilvános vagy privát felhőben. Linuxon futtatva a Docker a Linux kernel erőforrás-elszigetelési funkcióit (például cgroups és kernel névterek) és egy egyesítésre képes fájlrendszert használ, hogy a konténerek egyetlen Linux-példányon belül fussanak, elkerülve a virtuális gépek indításának és karbantartásának többletterheit. A macOS-en a Docker egy Linux virtuális gépet használ a konténerek futtatására.

Mivel a Docker konténerek könnyűek, egyetlen szerver vagy virtuális gép egyszerre több konténert is futtathat. Egy 2018-as elemzés szerint a Docker tipikus felhasználási esete nyolc konténer futtatását jelenti állomásonként, és az elemzett szervezetek negyede 18 vagy annál több konténert futtat állomásonként.

A Linux kernel névterek támogatása többnyire elszigeteli az alkalmazásnak az operációs környezetre vonatkozó nézetét, beleértve a folyamatfákat, a hálózatot, a felhasználói azonosítókat és a csatlakoztatott fájlrendszereket, míg a kernel cgroupjai a memória és a CPU erőforrás-korlátozását biztosítják. A 0.9-es verzió óta a Docker saját komponenst (az úgynevezett "libcontainer") tartalmaz a Linux kernel által közvetlenül biztosított virtualizációs eszközök használatához, amellett, hogy a libvirt, az LXC és a systemd-nspawn révén absztrahált virtualizációs interfészeket is használ.

A Docker egy magas szintű API-t valósít meg, hogy könnyű konténereket biztosítson, amelyek folyamatokat futtatnak elszigetelten. A Docker-konténerek szabványos folyamatok, így lehetséges a kernel funkcióit használni a végrehajtásuk megfigyelésére beleértve például az olyan eszközök használatát, mint a strace a rendszerhívások megfigyelésére és beavatkozására.

A Docker Compose egy eszköz több konténerből álló Docker-alkalmazások definiálására és futtatására.[29] YAML-fájlok segítségével konfigurálja az alkalmazás szolgáltatásait, és egyetlen paranccsal végzi el az összes konténer létrehozását és indítási folyamatát. A docker-compose CLI segédprogram lehetővé teszi a felhasználók számára, hogy egyszerre több konténeren futtassanak parancsokat, például képeket készítsenek, konténereket skálázzanak, leállított konténereket futtassanak, és így tovább.[30] A képmanipulációval kapcsolatos parancsok vagy a felhasználó interaktív opciói nem relevánsak a Docker Compose-ban, mivel azok egy konténerhez szólnak.[31] A docker-compose.yml fájl az alkalmazás szolgáltatásainak meghatározására szolgál, és különböző konfigurációs opciókat tartalmaz. Például a build opció olyan konfigurációs opciókat határoz meg, mint a Dockerfile elérési útja, a command opció lehetővé teszi az alapértelmezett Docker-parancsok felülírását, és így tovább. 2013. december 21-én jelent meg a Docker Compose első nyilvános béta verziója (0.0.1 verzió). 2014. október 16-án vált elérhetővé az első gyártásra kész verzió (1.0).[32] A Docker Compose első nyilvános béta verziója (0.0.1 verzió) 2014. október 16-án jelent meg.

## IdentityServer4

## SignalR

# Program részletes bemutatása

Leírni, hogy a bemutatás az úgy fog felépülni, hogy a backend aljától halad a frontending

## Backend

Beszélni a .net és asp.netről

### Mikroszolgáltatások

A dockerről részletesen írni, miért jó használni, mi az előnye, milyen szinten használja a programunk, docker-compose. A mikroszolgáltatásokról írni, hasonlóan milyen előnyökkel jár. Képet bevágni a docker compose egyik conténeréről.

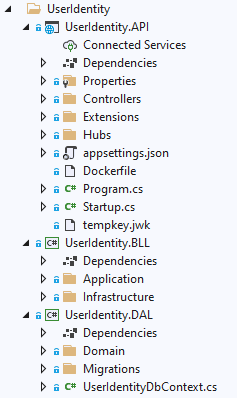
A képen szöveg látható

Automatikusan generált leírás

. ábra: Docker compose konténer

### Többrétegű architektúra, mappastruktúra

Nagyjából elővezetni, hogy milyen szinten lesznek kb szétosztva a fejezeteim, ugyanis a DAL, BLL, API rétegek mentén lesznek a nagyobb pontok. A rétegekről átfogóan beszélni, melyiknek mi a szerepe.



6. ábra: Rétegek, mappastruktúra

### Adatbázis

Elsőként az sql alapú adatbázisok előnyeiről/hátranyairól beszélni a nosql esekkel szemben, miért választottuk ezt. Majd a saját db-m részleteiről képek, felépítéséről.

### Adatelérési réteg

Elmondani, hogy ez az első szintje a 3 rétegnek, elmondani, hogy itt mi található.

#### Entity Framework

#### Code-First

#### DBContext

A képen szöveg látható

Automatikusan generált leírás

. ábra: DbContext kinézete

### Üzleti logikai réteg

Elmondani, hogy ez az második szintje a 3 rétegnek, elmondani, hogy itt mi található.

#### Command Query Responsibility Segregation minta

Beszélni a CQRS ről (mindegyiknél figyleni hogy elsőnek kiírni a teljes nevét), miért hasznos szétbontani commandokra és querykre a feladatokat.

#### Repository minta

Repositoryként működő storokról, ami egy plusz réteget ad, az egésznek.

#### Mediator minta

Mediatorról részletesen, ez szolgál a storeok és az api közötti résznek Itt zajlik a logika amiben nincs adatbázis művelet. Kép egy handle függvényről

#### Automapper

Szokásos bullshit, egyikről képet amiben van értelmes dolog is. (pl friend)

#### Globális kivételleképzés

Csak röviden a hellangról, egy saját kivételről kép is lehet.

#### Fontosabb logikai megoldások

Pl a friendek kezeléséről, lobby kezeléséről, inviteok kezeléséről lehet írni.

### Prezentációs réteg

Figyelni az API szó teljes kiírására elsőnek. Elmondani, hogy ez az harmadik szintje a 3 rétegnek, elmondani, hogy itt mi található.

#### Globális kivételkezelés

Csak kép a kódról a startupban és beszélni róla. A képen szöveg látható

Automatikusan generált leírás

. ábra: Saját kivételkezelés

#### WebSocket technológia

Elmondani mire jó, majd megmutatni pár sajátot.

#### Authentikáció

Az authentikáció flowjáról, tokenekről, konfigurációjáról.

#### Kontrollerek

Példa a mediator hívás indítására és elmondani, hogy a Commandok és Queryk micsodák.

#### Startup

Az elrendezésről is.

### API átjáró

Beszélni az ocelotról.

## Frontend

Beszélni az angularról, miért jó, miért ezt választottam hasonló.

### Komponsek

### Szervízek

### Pipe

### Guard

### Interceptor

### Snackbar

### WebSocket

# Összegzés, értékelés

## Továbbfejlesztési lehetőségek

Ahogy említettem, nagy fába vágtuk a fejszénket, de szerintem nagyrészt megvalósítottuk, amit előre elterveztünk. Azonban a program közel se tökéletes, rengeteg finomítani való lenne benne, vagy esetleg plusz funkciókkal tudna bővülni. A következő részben azt fogom felsorolni, hogy szerintem mi az, amitől használhatóbb lenne a játék és esetleg mivel lehetne továbbfejleszteni.

Ugye, mint említettem nulláról tanultam meg használni az Angulart de a HTML és a CSS részén is lenne hova fejlődnöm. Tehát maga a program kinézete, kezelőfelülete lehetne egységesebb és szebb. Photoshopban készültek is hozzá látványtervek, hogy mi az, amit szeretnék megvalósítani ám erre sajnos nem maradt elég idő, így a végső változatba se kerültek bele.

A program egy online játszható játéknak készült és az elején azt is terveztük, hogy majd kitelepítjük Azure-ba vagy valami külső helyre, hogy ki lehessen próbálni élesben több játékos által magát a játékot, ám ez sajnos elmaradt szintén az idő szűke miatt. Tehát továbbfejlesztésnek mindenféleképpen beleraknám, hogy fusson valami külső szerveren, hogy használható legyen mások által.

A programban megvalósult az eredményjelző is mind frontend mind backend részről. Ez megmutatja, az utolsó tíz játékod játszott karakterét és a helyezést, amit azzal értél el. Ám nem lett végül összekötve a társam oldalával, így a végén nem adja hozzá a listába az adott játékosoknak az elemet. Ilyen kis apróság még a váróból való játékos kirúgás. Ez elkészült backend oldalon azonban a frontenden sajnos figyelmetlenség miatt ez kimaradt a funkciók közül.

Authentikáció részről még akár bele kerülhetne egy kétlépcsős azonosítás is pluszba, és a mostani állapotán is lehetne finomítani.

## Értékelés

Összességében elégedett vagyok az elvégzett munka mennyiségével és minőségével. A technológiák többségét eddig csak felületesen vagy egyáltalán nem ismertem, ugyanis az Angularral előtte minimálisan se foglalkoztam, így azt nulláról kellett megtanulnom használni. A backend oldali technológiák egy részét ismertem már és használtam korábban is azonban voltak, amiket csak felületesen ismertem. Így a szakdolgozat elkészítése nagy mértékben hozzájárult a szakmai fejlődésemhez.

A csapatmunka is gördülékenyen zajlott. Mint ahogy azt korábban említettem megpróbáltuk minél jobban elszeparálni a feladatokat így a fejlesztés alatt elég volt minimális szintű kommunikáció. Az elején a tervezést részénél kellett többet egyeztetni, megbeszélni. A fejlesztés alatt leginkább az elkészítendő kártyák képei miatt volt interakció. A fejlesztés vége felé kapcsoltuk össze a játék és a menü részét, ezért itt is szükséges volt a kommunikáció.

Irodalomjegyzék

1. Wikipedia, „List of most expensive video games to develop” [Online]. Available: <https://en.wikipedia.org/wiki/List_of_most_expensive_video_games_to_develop>
2. Wikipedia „Adobe Flash” [Online]. Available: <https://hu.wikipedia.org/wiki/Adobe_Flash>
3. Wikipedia „Docker (software)” [Online]. Available: <https://en.wikipedia.org/wiki/Docker_(software)>

Függelék