

Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Markovics Gergely

**Kártyajáték  
megvalósítása mikroszolgáltatás alapokon**

Konzulens

Dr. Kővári Bence András

BUDAPEST, 2023

Tartalom

[Összefoglaló 6](#_Toc148304524)

[Abstract 7](#_Toc148304525)

[1 Bevezetés 8](#_Toc148304526)

[1.1 Böngészős- és társasjátékok 8](#_Toc148304527)

[1.2 Motiváció 8](#_Toc148304528)

[1.3 Sushi Go Party! 9](#_Toc148304529)

[1.4 Felhasználói élmény 10](#_Toc148304530)

[1.5 Megvalósítás 10](#_Toc148304531)

[2 Technológia 12](#_Toc148304532)

[2.1 Bevezetés 12](#_Toc148304533)

[2.2 Szerveroldali technológiák 12](#_Toc148304534)

[2.2.1 Docker 12](#_Toc148304535)

[2.2.2 ASP.NET CORE 13](#_Toc148304536)

[2.2.3 Ocelot 13](#_Toc148304537)

[2.2.4 MediatR 14](#_Toc148304538)

[2.2.5 SignalR 14](#_Toc148304539)

[2.2.6 Redis 15](#_Toc148304540)

[2.2.7 RabbitMQ 15](#_Toc148304541)

[2.2.8 IdentityServer4 16](#_Toc148304542)

[2.2.9 AutoMapper 16](#_Toc148304543)

[2.2.10 Hangfire 16](#_Toc148304544)

[2.2.11 xUnit 16](#_Toc148304545)

[2.3 Kliensoldali technológiák 16](#_Toc148304546)

[2.3.1 Angular Material 17](#_Toc148304547)

[2.3.2 SCSS 17](#_Toc148304548)

[2.3.3 SignalR 17](#_Toc148304549)

[2.3.4 NGX-Translate 18](#_Toc148304550)

[2.3.5 Dotenv 18](#_Toc148304551)

[2.3.6 Cypress 18](#_Toc148304552)

[3 Tervezés 19](#_Toc148304553)

[3.1 Bevezetés 19](#_Toc148304554)

[3.1.1 Felhasználókezelés 19](#_Toc148304555)

[3.1.2 Bolt rendszer 19](#_Toc148304556)

[3.1.3 Váróterem rendszer 20](#_Toc148304557)

[3.1.4 Játék rendszer 21](#_Toc148304558)

[3.2 Architektúra 22](#_Toc148304559)

[3.2.1 Áttekintő 22](#_Toc148304560)

[4 Önálló munka bemutatása 23](#_Toc148304561)

[4.1 Bevezetés 23](#_Toc148304562)

[4.2 Szerveroldali funkciók 23](#_Toc148304563)

[4.3 Kliensoldali funkciók 23](#_Toc148304564)

[4.4 Felhasználókezelés 23](#_Toc148304565)

[4.5 Bolt kialakítása 23](#_Toc148304566)

[4.6 Váróterem megvalósítása 23](#_Toc148304567)

[4.7 Játék 23](#_Toc148304568)

[5 Összefoglaló 24](#_Toc148304569)

[6 Irodalomjegyzék 25](#_Toc148304570)

[7 Függelék 26](#_Toc148304571)

Hallgatói nyilatkozat

Alulírott **Markovics Gergely**, szigorló hallgató kijelentem, hogy ezt a diplomatervet 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, 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, 2023. 05. 28.

...…………………………………………….

Markovics Gergely

Összefoglaló

A kártyajátékok régóta nyújtanak szórakoztatást és kikapcsolódást világszerte. Legyen az családi vagy baráti környezetben. Könnyed közösségi élményt biztosítanak minden korosztály számára. Ezen játékok közül a Sushi Go egy komplex, változatos és egyedi játékélményt ad az összegyűlt társaságnak.

A kártyajátékok egyik hátránya, hogy fizikai jelenlétet igényelnek a játékosoktól. Legyen akármilyen élvezetes a játék, a játékosoknak össze kell gyűlniük a használatához, ami a mai világban gyakran nehezen megoldható. Ennek a problémának az áthidalására szolgál a játék szoftveres megvalósítása, amivel a játékosok globálisan, akárhonnan tudják élvezni a játék adta lehetőségeket.

A játék szoftveres implementációja böngészőben játszható formában készül el, aminek a segítségével a játékosok platformtól függetlenül csatlakozhatnak be a játékba, élvezhetik annak szolgáltatásait. Azért, hogy igazi közösségi felhasználói élményt nyújtson az alkalmazás, a fejlesztés során implementálva lettek a játékon túlmutató funkciók, mint barátok kezelése, vásárlás vagy szöveges beszélgetés nyújtása.

A szoftver implementációjában ezenkívül kiemelt szerepet kapott, hogy a szerveroldali komponens mikroszolgáltatás alapokon működjön, azt aktívan és hatékonyan kihasználva.

A feladat tehát egy ismert kártyajátékot implementáló szoftver megvalósítása webes klienssel és mikroszolgáltatásokra épülő szerverrel, figyelmet fordítva a felhasználói igényekre és élményre.

Abstract

Card games have long provided entertainment and relaxation around the world. Whether with family or friends, they provide a light social experience for all ages. Of these games, Sushi Go provides a complex, varied and unique gaming experience for socialising.

One disadvantage of card games is that they require physical presence. No matter how enjoyable the game, players need to come together to use it, which is often difficult in today's world. The software implementation of the game is a way to overcome this problem, allowing players to enjoy the game's features globally.

The software implementation of the game will be in a browser-based format, allowing players to connect to the game and enjoy its features regardless of platform. In order to provide a true social user experience, features beyond the game, such as managing friends, shopping and text chat, were implemented during development.

In addition, the implementation of the software has focused on ensuring that the server-side component is microservice based, actively and efficiently exploiting the server-side component.

The task is therefore to implement a software implementing a well-known card game with a web client and a server based on microservices, paying attention to user needs and experience.

# Bevezetés

## Böngészős- és társasjátékok

Társasjátékok évszázadok óta nyújtanak szórakoztatást és kikapcsolódást családoknak és barátoki társaságoknak. Ezek az interaktív, változatos játékok nem csak szórakoztatást nyújtanak, de különböző készségeket is fejlesztenek, mint a kommunikáció, vagy stratégiai és logikai gondolkodás.

Fizikai társasjátékokkal ellentétben a böngészős játékok csak az internet korával jelentek meg, és azóta rohamosan nő a népszerűségük. A folyamatosan frissülő és fejlődő felület sok figyelmet és érdeklődést vonz maga után, legyen az több- vagy egyszemélyes játék. Ezen játékokhoz a hozzáférés általában teljesen vagy részlegesen ingyenes, így szélesebb közösséget tudnak kialakítani.

Böngészős játékok további előnyei, hogy könnyű hozzáférést biztosítanak, mind a használatuk és időigényük rugalmas, emellett könnyed közösségi interakcióra is adnak lehetőséget. Kényelmes a használatuk platformtól függetlenül, legyen az komplett asztali számítógép, laptop, vagy kisebb mobil vagy tablet. Ezek lehetőséget biztosítanak arra, hogy igényektől függően bárhol és bármikor élvezhessük a játékot.

A dolgozat témájában a megvalósítás során tehát a kettőnek a közös pozitív tulajdonságaikra összpontosítottam. Tehát egy könnyen elérhető és közösségi élményt nyújtó, de emellett változatos játék megvalósítása.

## Motiváció

Ez a diplomaterv a Sushi Go Party! nevű változatos kártyajátékot hivatott megvalósítani interneten keresztül. Az interneten keresztül való használatával a játéknak nagyobb célközönséget tud teremteni magának.

A szoftver elkészítésének egyik alap motivációja, hogy egy teljeskörű szoftvert akartam létrehozni, ami egyszerre tartalmaz szerveroldali logikát, és kliensoldali dizájn megvalósítást is. Ehhez a feladathoz egy kártyajáték implementációját gondoltam megfelelőnek, mivel azontúl, hogy könnyen megfogalmazható milyen végterméket szeretnék megvalósítani, sokféle kreatív bővítési lehetőséget biztosít.

A szoftverhez használt kártyajátéknak a „Sushi Go Party” -t választottam, mivel egyedi és izgalmas játékelemeket tartalmaz, ami a játékosoktól a játékhoz illő egyedi stratégiákat vár el. Emellett nem csak maguk a lapok és használatuk egyedi, de maga a pakli is sokféleképpen változtatható a játékosok idejétől, számától, vagy csak kedvétől függően. Az ilyen részletesség lehetővé teszi a szoftverben további konfigurációs, közösségi vagy gazdasági lehetőségeket is.

## Sushi Go Party!

A „Sushi Go Party!” kártyajáték a „Sushi Go” nevű kártyajáték második verziója, további lapokkal és többféle pakli összerakásának lehetőségével kiegészítve.

A játék szabálya röviden leírva úgy hangzik, hogy először minden játékos a játék elején kap a kezébe egy kártyacsomagot különböző japán konyhát idéző ételek kártyáival. Az egyes kártyáknak különböző képességeik vannak, de a legfőbb céljuk, hogy valamilyen módon pontot szerezzen velük a játékos.

A játékban úgy zajlik egy kör, hogy minden játékos kiválaszt egy lapot titokban, és kirakja maga elé az asztalra. Ha mindenki kiválasztott egyet, akkor egyszerre felfordítják. Itt az egyes lapoknak más-más képességüktől függően vagy simán pontot érve lent maradnak a játékos előtt, vagy további akciót hajtanak végre. Ha ezek lezajlottak, minden játékos továbbadja a kezében lévő lapokat a mellette ülőnek, és kezdődik a következő kör. Ez a mechanika adja a játéknak az egyik stratégiai egyediségét.

A játékosok célja, hogy a kihasznált lapokkal pontokat gyűjtsenek, és a játék végén velük a legtöbbet szerezzék meg. Például a nigiri kártyák annyi pontot érnek, amennyi szerepel rajtuk, további logika nélkül. Viszont a tempura esetén minden második lerakott után lehet pontokat szerezni.

Egy körnek akkor van vége, ha minden lap elfogyott a játékosok kezéből. Vannak kártyák, amiknek a játéknak ennek a szakaszában van akciójuk. A kör végén, hacsak nem ilyen akciójú (desszert típusú) a lap, minden lerakott kártya visszakerül a pakliba a pontja beszámítása után. Egy játék 3 ilyen körből áll, és az utolsó kör végén a legtöbb összesített ponttal rendelkező játékos nyer.

A játékot lehet játszani akár csak ketten is, amire van külön kifejezett erre kitalált pakli, és akár nyolcan is, amire ugyanúgy van sok játékosra kifejlesztett kártya kombináció. Az aranyos rajzokkal ellátott lapokon egyértelműen megtalálhatjuk melyik lapnak mi a képessége, így fiatalok és öregek is könnyedén betanulhatják a játékot, könnyed családi program keretében is.

## Felhasználói élmény

A feladat keretében a hangsúly nemcsak a kártyajáték implementációjára összpontosult, hanem a felhasználói élmény és a vele járó segédfunkciók megvalósítására is. Tehát a szoftverben fontos szerepet játszik, hogy közösségi elemeket, funkciókat is szolgáltasson a felhasználói számára.

A játékban tehát a felhasználókezelési funkciók és annak megvalósítása is fontos volt. Az egyes felhasználóknak regisztrálniuk kell, hogy igénybe vegyék a játék adta funkciókat és azon felüli lehetőségeket is igénybe vegyenek. Ilyen például, hogy egymást barátnak tudják jelölni, amivel megtalálják egymást könnyedén a rendszerben.

Ezenkívül, hogy kihasználjuk azt a helyzetet, hogy többfajta paklilehetőség van, egyfajta belső bolt is üzemel a honlapon. Ebben a boltban a játékból szerzett pontokkal tudnak a játékosok beváltani egyes paklikat, amikkel játszani szeretnének. Ezzel is növelve a játékélményt, motivációt.

A játékhoz értelemszerűen szükség van egy váróteremrendszerre is, ahol a játékosok a játék létrehozásánál megtalálhatják egymást. Itt játszik szerepet a megvásárolt paklik jelentősége is. Az ilyen termekben szabadon tudnak szövegesen csevegni egymással a játékosok, míg a játék indítására várakoznak.

További felhasználói élményt segítő funkciók is megtalálhatóak a szoftverben, mint a honlapon beállítható többnyelvűség, vagy sötét és világos stílus közötti választási lehetőség.

## Megvalósítás

A projekt részeként megvalósult egy szerveroldali és egy kliensoldali alkalmazás. A szerveroldali komponens implementálásánál fontos tényező volt, hogy az mikroszolgáltatás technológia alapjain legyen megvalósítva.

A szerveroldali komponens a főbb funkciók mentén mikroszolgáltatásokra lett osztva. Ezáltal elhatárolva egymástól elemeit a képességeik, felelősségeik mentén, közöttük elvágva a kapcsolatot. Ezek a főbb funkciók a felhasználó és barátkezelés, a boltnak a kezelése, a váróterem és beszélgetésnek a kezelése, és magának a játéknak a kezelése. Ezek a funkciók között laza kapcsolat van, és csak ritkán van igény közöttük való kommunikációra. Ez funkcionálisan be van építve a szerverbe, mint konténerek közötti kommunikáció.

A szerveroldali mikroszolgáltatások használatánál fő szempont volt, hogy a felosztottság a külvilágból transzparens legyen, tehát a szerver egyetlen kapun keresztül kommunikáljon bárkivel. Ebbe beletartozik a WebSocket kommunikáció is, és a statikus adatok, mint például a felhasználók profilképeinek, vagy egyéb képeknek a linkjei is.

Szerveroldalon ezenkívül a perzisztens relációs adatbázis mellett egy temporális cache-elésre használt Redis alapú cache adatbázis is üzemeltetve van, ami sokban növeli a szerver teljesítményét, gyorsaságát. Ezzel tovább növelve a felhasználói élményt.

Emellett a szerver a szokványos REST alapú kommunikáción felül lehetőséget biztosít WebSocket technológiára épülő kommunikációval való csatlakozást is, amivel élőben értesülhetünk kliensoldalon az egyes eseményekről. Például, ha egy váróteremben egy barátunk üzenetet küldött.

Kliensoldalon kiemelt figyelmet fordítottam az igényes, átlátható dizájnra. A kliens feladata a kommunikáció a szerverrel, mind REST mind WebSocket formában. Az utóbbinál kezelnie kell, hogy melyik oldalakat megtekintve melyik kapcsolatot tartsa fenn, és hogy az egyes kapcsolatok eseményeivel milyen műveleteket hajtson végre.

Kliens szerepe ezenkívül a többnyelvűség és stílus megvalósítása is. A kiválasztott nyelvet és stílust egy-egy Cookie segítségével tárolom, hogy az oldal frissítésekor, vagy újra megnyitásakor is a kedvenc kiválasztott segítségével jelenjen meg.

Egyelőre két nyelven, magyarul és angolul jelenik meg a honlap, de végtelen bővítési lehetőség van. A stílusokból is kettőt választhatunk, egyet, ami a honlap világos témáját teszi ki, egyet meg ami a sötétet. Viszont itt is könnyedén tudjuk bővíteni további témákkal.

Ezenkívül, hogy a kommunikációhoz szükséges szenzitív információk biztonságos helyen legyenek tárolhatóak, az egyes kulcsok nem a környezeti változók között vannak tárolva, hanem script segítségével töltődnek be oda indításkor.

# Technológia

## Bevezetés

Az elkészült szoftver számos és változatos technológiai skálát tartalmaz, kezdve az adatbázistól a böngésző stílusáig. Ebben a fejezetben pár főbb technológiát sorolok fel, amik meghatározóak voltak a szoftver fejlesztése során.

Nagy befolyással voltak rám mind a BSc, mind az MSc alatt elkészített korábbi munkáim, amikből ihletet és tudást merítettem. Az elkészült alkalmazás tekinthető technológiai szempontból a korábbi munkáim továbbfejlesztésének, mivel az alattuk megismert technológiák részletesebb megismerése és használata volt a szoftver célja.

## Szerveroldali technológiák

A szoftver fejlesztése során szerveroldalon számos technológia volt felhasználva. Ezeknek elsődleges feladata a mikroszolgáltatások kialakítása és kommunikációja, emellett bennük a háromrétegű architektúra kialakításának megvalósítása volt.

### Docker

A Docker [1] egy olyan nyílt forráskódú technológia, ami lehetővé teszi alkalmazások gyors és egyszerű csomagolását, konfigurációját egy környezetfüggetlen környezetben. Tehát egy konténerizációs platform, ami operációs rendszer szintű virtualizációt biztosít.

Segítségével alkalmazások és függőségeik környezettől függetlenül, szabadon futtathatóak, mivel egységesen csomagolva alakítja ki hordozhatóságukat. Az így kialakított konténerek el vannak szeparálva egymástól, ezáltal beállításaik és függőségeik is külön-külön kezelhetőek redundancia nélkül, átláthatóan. Az egyes konténerekre API felületet biztosít, így könnyedén megfigyelhetjük a belső folyamatokat.

A létrejött konténerizált rendszer könnyűsúlyú, jól skálázódik a szerver létrehozásánál. A rendszer kialakításához én a Docker Compose segítségét használtam, ami egy YAML konfigurációs fájl segítségével engedi, hogy megfogalmazzam az egyes konténereket és a hozzájuk tartozó paramétereket, környezeti változókat.

A Docker tehát megkönnyíti a fejlesztők és szakemberek feladatát a fejlesztésben, telepítésben és a komponensek skálázásában is. Hátránya viszont, hogy a fejlesztett kódban redundanciát okozhat, ha az egyes konténerekben hasonló felelősségek vannak. Figyelni kell a kialakítás során, hogy megfelelően válasszuk el a konténerek funkcióinak a határát, esetleg segédkomponensekkel támogatva. Emellett meg kell valósítani a konténerek közötti esetleg kommunikációnak a lehetőségét, ami az izolált rendszerek közötti logikai kapcsolatot hozza létre.

### ASP.NET CORE

Az ASP.NET Core [2] egy open-source, cross-platform keretrendszer, amit a Microsoft fejlesztett ki webes alkalmazások és API-k készítésére. A szoftverhez én a .NET 7-es verziójú változatát használtam, ami dolgozat írása alatt a legújabbnak számított. A keretrendszer lehetővé teszi fejlesztők számára hatékony és skálázható szoftverek fejlesztését. Dockeres technológiákkal is széleskörű támogatás található benne. Tehát egy megbízható és hatékony keretrendszer szerveroldali alkalmazások fejlesztésére, ami webes alkalmazások hátterének könnyen használható.

A keretrendszer segítségével létrehozott alkalmazásokban könnyedén tudunk további segédkönyvtárakat telepíteni „NuGet package” -ekként, ami ki is volt használva a dolgozatban. Emellett beépített támogatást tartalmaz az egyes szolgáltatások injektálására, amivel rendezett, átlátható struktúrát tudtam létrehozni a fejlesztés során.

### Ocelot

Az Ocelot [3] egy olyan keretrendszer, segédkönyvtár, aminek elsődleges célja, hogy .NET technológiával futó mikroszolgáltatások fölé egy kívülről transzparens, egységes felületet biztosítson. Ez a közös felületet biztosító komponenst az „API Gateway” a szerverben.

Az Ocelot az üzenetek átirányítása során többféle stratégiát vagy transzformációt alkalmazhat. Emellett átlátható integrációt valósít meg az „IdentityServer” -rel, ami a szerver felhasználókezelő komponense. Ezáltal könnyedén lekezeli a felhasználók authentikációját, ha a helyzet úgy adja.

A Gateway-en keresztül nem csak a REST alapú kérések, de a WebSocket technológiára épülő kétirányú kommunikáció is támogatva van, elősegítve a szerverben való implementálását is.

Az Ocelot tehát egy hatékony és rugalmas könyvtár konténerizált rendszerek elrejtésére egy API gateway mögé, ezzel megkönnyítve a szerver külső használatát, esetleg elrejtve a nem publikálandó belső működést. Ezzel bizonyos szinten biztonságot is nyújtva.

### MediatR

MediatR [4] egy C#-hoz készült segédkönyvtár, ami a mediátor minta megvalósítását hivatott segíteni a fejlesztésben. Használata leegyszerűsíti a kérések és parancsok regisztrálását és a megvalósításuk delegálását a megfelelő komponensek felé. Segíti a szoftverben a kód jobb széttagoltságát és a felelősségek megfelelő elválasztását. Laza csatolás biztosításával a funkciók jobb skálázhatóságát is biztosítja.

A könyvtár elég rugalmasan és egyszerűen használható a beépített osztályok segítségével, továbbá a WebSocket események lekezeléséhez is ad további segítséget. Lehetőséget ad a CQRS minta implementálására, tehát a parancsok és lekérdezések szétválasztására. Emellett lehetőséget ad az esemény feliratkozás minta használatára is, ami a cache adatbázis naprakészen tartásában adott segítséget.

### SignalR

A SignalR [5] egy open-source segédkönyvtár, ami leegyszerűsíti a való idejű kommunikáció megvalósítását ASP.NET Core alkalmazásokban. A keretrendszer beépített eleme, így könnyen felhasználható vagy kombinálható más komponensekkel, mint például az authentikáció kezelése.

Az alkalmazás szerves és fontos része volt a kétoldalú kommunikáció megvalósítása, mivel a megvalósított funkcióknál nagy szerepe volt, hogy a felhasználók élőben, rögtön értesüljenek a változásról, ne kelljen manuálisan nekik lekérdezniük.

A SignalR rugalmasan használható, megfogalmazhatunk számos kapcsolatot, amiknek megszabhatjuk, hogy mi a kommunikációs interfésze a klienssel. Emellett a kapcsolatok dinamikus kezelésével, csoportosításával nyomon tudjuk követni az élő kapcsolatokat. A kialakított csoportok segítségével nem csak egy-egy tudunk üzeneteket váltani, hanem jól definiált csoportosítások mentén broadcast is tudunk értesítéseket küldeni a megfelelő felhasználóknak. Erre egy példa, ha egy játékos kirak egy lapot, akkor arról csak az aktuális játék résztvevői értesüljenek.

Mivel az egyes konténerek el vannak zárva a külvilágtól, ezért a megvalósított WebSocket kommunikáció nem közvetlenül a klienssel zajlik le, hanem az Ocelot feladata elosztani a megfelelő végpontok között ki kivel kommunikál. Szerencsére az Ocelot lehetőséget ad nem csak REST alapú kommunikációra. Az így kialakított átirányítást továbbá authentikáció is védi, így magasabb védelmet biztosítva az alkalmazás felhasználói számára.

### Redis

Redis [6] egy gyors és könnyen skálázható kulcs-érték alapú adatbázis, amit a gyorsasága miatt cache-elés megvalósítására használtam a kérések megvalósítása során. A Redis egy in-memory típusú adatbázis, tehát minden adatot a memóriában tárol el. Ez adja a rendkívüli gyorsaságát és hatékonyságát, ami miatt gyakran használják. Emellett egyszerű és rugalmas integrációt biztosít a konténerizált és .NET alapú rendszereknek, amit én is kihasználtam.

Gyakorlatban elsősorban a MediatR lekérdezések és parancsok környékén volt szerepe. Bizonyos lekérdezéseknél be lett állítva, hogy a lekérdezés tovább küldése helyett először ellenőrizze le, hogy a megadott kulcshoz volt-e már elmentett érték a cache adatbázisban. Ha talált ott értéket, akkor a lekérdezés lefuttatása nélkül visszatért. Az ilyen kulcs-érték párok frissítését vagy a MediatR esemény feliratkozás lehetőségével oldottam meg, vagy egyes parancsok visszatérését használtam fel.

### RabbitMQ

RabbitMQ [7] rendszer elsődleges feladata a konténerizált rendszerben a konténerek közötti kommunikáció megvalósítása volt. A RabbitMQ egy message queue rendszer, ami a megvalósított mintában a message broker szerepet tölti be. Itt az egyes konténer komponensek felvehetik a kibocsátó és fogadó szerepet, elküldve az üzeneteket a feliratkozó konténerek felé, és fogadva a feliratkozott üzeneteket a küldő konténerek felől.

Mivel az egyes felelősségeket nem lehet teljesen elválasztani egymástól, ezért gondoskodni kell arról, hogy az egyes konténerek tudjanak egymással kommunikálni amikor szükséges. Itt van szerepe a RabbitMQ-nak, hogy bizonyos eseményekre parancsokat futtassunk le más-más konténerekben, mint amiken meghívtuk a kéréseket.

### IdentityServer4

Az IdentityServer4 [8] szerepe volt az alkalmazásban a felhasználói műveletek kezelése. Ebbe beletartozik a regisztráció, belépés vagy esetleg törlés vagy módosítási műveletek is. Nagy szerepet töltött be mind a szerver mind a kliens fejlesztésében.

Maga az IdentityServer4 egy felhasználókezelő konténerbe lett beépítve, ahonnan többféle jogosultság, token kezelés vagy authentikációs és authorizációs szerepet lát el. A kliensnek bejelentkezés során átadott tokeneket sokszínűen tudjuk testre szabni, így a szükséges jogosultságokat átadni a weboldalnak a bejelentkezett felhasználóról.

Az OAuth2 technológiára épülő authentikációt széles körökben használják, és az IdentityServer4 is ad rá lehetőséget, hogy a szerverünk felhasználóit ezzel a technológiával védjük meg.

Emellett egyszerű integrációt biztosít a szerver többi komponensével, mint az Ocelot vagy SignalR könyvtárak.

### AutoMapper

Az AutoMapper egy egyszerű segédkönyvtár, ami az egyes domain osztályok átfordításában segít vagy adatátviteli objektumokra, vagy nézetmodellekre. Könnyen kezelhető, de jól testre szabható profilok megfogalmazásával.

Használatával átláthatóbban tudjuk kezelni a bemenetek és kimenetek feldolgozását és kialakítását anélkül, hogy tele szemetelnénk a lekérdezéseket vagy adatosztályokat. Egyszerű támogatás is biztosítva van a használatára.

### Hangfire

A Hangfire [9] egy olyan segédkönyvtár, ami lehetővé teszi .NET alapú rendszerekben háttérben futó és ismétlődő események kezelését, létrehozását vagy törlését. Széleskörű konfigurációs lehetőségekkel, perzisztens adatbázissal könnyen megfogalmazhatunk olyan folyamatokat, amiket felhasználói események nélkül, vagy azoknak a hatására adott idővel később akarunk végrehajtani.

Emellett lehetőséget ad arra, hogy az időzített folyamatok során a CQRS mintában megfogalmazott parancsokat hajtsa végre a megadott paramétereket JSON-be konvertálva tárolva.

### xUnit

Egy tesztelési eszköz, ami .NET alapú rendszerekben Unit tesztek megfogalmazására ad lehetőséget. Különálló tesztelési projektekben megfogalmazva a Visual Studio segítségével tudjuk kezelni a velük létrehozott teszt függvényeket. Így a Moq könyvtár segítségét is felhasználva átlátható Unit teszteket tudunk létrehozni, amiket a fejlesztőkörnyezetből követhetünk milyen eredményt adnak.

## Kliensoldali technológiák

Kliensoldali technológiák főleg a dizájn kialakítására fókuszálnak. Használatuk az egységes, átlátható felület kialakítását vagy a szerver felé irányított kapcsolat megvalósítását segítik elő.

### Angular Material

Az Angular Material [9] az Angular keretrendszer által kifejlesztett UI komponenskönyvtár. Elsődleges célja, hogy egy egységes és esztétikus dizájnt adjon a felületnek, amit professzionális környezetben is gyakorian használnak fel. A könyvtár számos beépített funkciót és komponenst szolgáltat, amelyeknek a segítségével a legtöbb elemet ki lehet alakítani, amire egy honlapon szükség lehet. Ezekkel a komponensekkel gyorsan, egyszerűen létrehozhatjuk a kliensoldali alkalmazásunkat a mély testreszabhatóságuk segítségével. A dokumentációját böngészve könnyen megtaláljuk a nekünk szükséges elemeket az interfészükkel és változatos példákkal együtt. Egy gyorsan fejlődő és változó keretrendszerről van szó, de a verziók között is átláthatóan tudunk barangolni.

A világos és sötét stílus kialakításánál is nagy szerepet játszott az Angular Material, mivel a számos beépített komponens stílusával összhangban volt szükség a beállítására. Szerencsére a könyvtár a stílus szerkesztésére is ad konfigurációs lehetőséget.

### SCSS

A CSS kiterjesztése, a stílus kialakításában volt fő szerepe. Az alap CSS funkciókon felül számos további lehetőséget biztosít a stílus fejlesztésében. Kialakíthatunk vele hierarchikus, átlátható stílusfákat, bevezethetünk később felhasználandó változókat, vagy további kiegészítő függvényeket írhatunk bele. A változókezelésnek nagy haszna volt a világos és sötét stílus kialakításában is.

Segítségével változókba lehet szervezni az Angular Material által adott és beállított színsémát. Így csökkentve az erőforrásigényét az egyes stílus fájloknak, mivel csak ezekre a színekre kell hivatkozniuk.

### SignalR

Mivel a WebSocket technológia kétoldalú kommunikáció, ezért meg kell említeni a kliensoldali felhasználását is. Angular felől az elsődleges feladata a kapcsolatfelépítés kérése a szerver felé a már említett authentikáció felhasználásával is.

Kliensoldalon arra kell figyelni a használatakor, hogy milyen esetekben akarjuk inicializálni és leállítani a kapcsolatokat. Milyen oldalak megtekintésekor van szükségünk milyen kapcsolatra. Ezenkívül, a már kialakított kapcsolaton lehetőséget biztosít a szerver felől érkező eseményekre való feliratkozásra, az onnan érkező adatok kezelésére.

### NGX-Translate

Az ngx-translate [10] egy erőteljes internalizációs könyvtár Angularra, amivel egyszerűen és jól skálázhatóan tudjuk kezelni a kliensoldali többnyelvűséget. A könyvtár segítségével a statikus, honlapon megjelenítendő szövegeket nem közvetlenül égetjük az alkalmazásba, hanem kiszervezzük őket Asset fájlokba, ahol a könyvtárban beállított nyelv segítségével választja ki melyik Asset segítségével állítsa be.

A könyvtárnak több más segédfunkciója van, ami sokban elősegíti a segítségével a többnyelvű fejlesztést. Például nem csak statikus szöveget tud többnyelvesíteni, hanem paraméterezett értékeket is tud kiértékelni a megjelenítés előtt. Ezzel például a névsorrendet is az adott nyelv szerint tudjuk megjeleníteni, de mondatoknak a tárgyai is ugyanígy könnyen megadható kívülről.

### Dotenv

A dotenv egy hasznos és népszerű, mégis egyszerű könyvtár, aminek a segítségével biztonságosan tudjuk kezelni a kliensoldali szenzitív adatainkat anélkül, hogy azokat kiadnánk a külvilág felé. Ha fejlesztők külön-külön dolgoznak kliens vagy szerveroldali alkalmazáson, akkor gyakran előjön a probléma, hogy kulcsokat és titkokat kell megosztaniuk egymás között. Ezeket nem akarjuk a Git repository-ban nyilvánosságra hozni, mivel a jövőben fel is használhatják ellenünk. Emellett az Asset-ek közé se akarjuk őket helyezni, mert akkor bárki kiolvashatja kívülről.

Ilyenkor jön szerepbe a dotenv könyvtár, ami egy olyan konfigurációs fájlból szolgáltatja az értékeket, ami nem része a Git repository-nak, de futás közben kiolvashatóak az értékek belőle.

Jelen szoftverben olyan felhasználási módszert valósítottam meg, ami a Dotenv konfigurációs értékeket a kliens inicializálásakor egy script segítségével kiolvassa, és az Angularhoz használható tárolóba tölti be őket. Így amellett, hogy nincsenek a Git irányítása alatt az értékek, a kliens is könnyedén használni tudja a megszokott módon.

### Cypress

# Tervezés

## Bevezetés

Az elkészített alkalmazás egy komplex, különálló egységekre bontható szoftver implementációja volt. Ezek a funkciók mentén feldarabolható az alkalmazás specifikációja is.

### Felhasználókezelés

A webalkalmazás felhasználókat kezel, hogy személyre szabott játék élményt tudjon nyújtson a játékosoknak. Elsősorban azért van rá szükség, mivel sok olyan funkciót valósít meg az alkalmazás, ami személyre szabott előzményeket és állapotot igényel, így be kell azonosítani ki szeretné használni a weboldalt.



1. ábra Felhasználókezelési használati esetek

Emiatt az oldalra látogatva az első, amit egy megtekintő megtehet, hogy vagy belép, vagy regisztrál. Enélkül nem mehet tovább az oldalon semerre. Egy felhasználótól adatokat gyűjtünk, ami a neve, felhasználóneve, e-mail címe és egy biztonságos jelszava. Ezek az adatok validálva vannak eltárolva. Ezenkívül a játékosok bejelentkezés után profilképet is állíthatnak maguknak, ami megjelenik a váróteremben és a játékban is.

További funkció a felhasználókezelésben a barátok kezelése. A felhasználók bejelölhetnek más felhasználókat barátnak, és azokat a kéréseket elfogadhatják vagy elutasíthatják. Barátok látják egymást egy listán, mellettük egy indikátorral jelezve éppen aktívak-e a honlapon. Ez elősegíti, hogy a játékosok megtalálják egymást.

A felhasználók ezenkívül tudják módosítani a felhasználójukat, akár a nevüket vagy felhasználónevüket is. Emellett a honlap megtekintése közben változtathatják a stílus vagy nyelv preferenciájukat is, amit egy Cookie-ba elmentve megjegyez.

### Bolt rendszer

A webalkalmazás tartalmaz egy beépített boltot, amiben a játékosok a játékok végén szerzett pontokból tudnak vásárolni.

Maga a megvalósított játék úgy épül fel, hogy a játék kezdete előtt ki kell választani egy pakli összeállítást, aminek a kártyáit szeretnénk felhasználni. Itt jön képbe a boltrendszer, hogy ne adjuk oda a játékosoknak a teljes összeállítási listát, hanem az alkalmazás használatával tudjanak tetszőlegesen felnyitni új módokat.

A képen szöveg, képernyőkép, hold, kör látható

Automatikusan generált leírás

2. ábra Boltrendszer használati esetei

Ezzel további motivációt adunk a játékosoknak, hogy többet használják az alkalmazást. Emellett nagyobb változatosságot is ad, meg döntési lehetőséget is, hogy milyen pakli tetszett meg nekik.

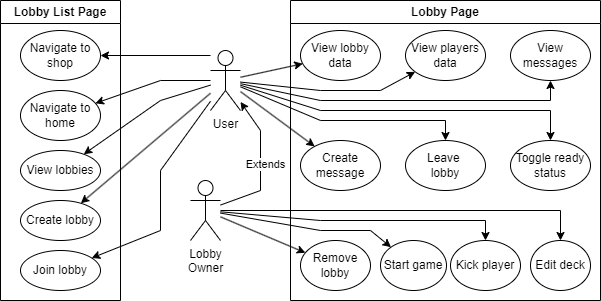
Minden játékos a játék alap Sushi Go verziójának paklijával tud kezdetben játszani, és elég pontot szerezve tudja magának feloldani a boltban vásárlási lehetőséget. Ezzel kicsit elrejtve a plusz ponthasználat szükségességét, de mégis újdonságot adva.

### Váróterem rendszer

Az alkalmazás várótermeket, „lobby” -kat kezel. Ennek a rendszernek a segítségével tudják a felhasználók megtalálni egymást egyes játékokra.

A várótermek komplex funkciókkal lettek ellátva. Először is megtekinthetjük az aktív várótermek listáját, aminek az aktuális állapotát élőben látják változni a honlap megtekinthetői. Tehát ha valaki elindít egy szobát, akkor azt egy másik felhasználó rögtön lát a lista változásával. Ugyanez fordítva, ha egy szoba megszűnik, akkor arról minden megtekintő rögtön értesül.

Egy szobát névvel és jelszóval tudunk létrehozni, amibe a játékosok a jelszó segítségével tudnak belépni. Ezzel valamelyest levédve a szobát, hogy csak azok léphessenek be, akiknek elárultuk a jelszót.



3. ábra Váróterem használati esetei

A szobába lépve a lehetséges funkciók aszerint változnak, hogy a felhasználó hozta-e létre, vagy csak utólag belépett. A szobában többféle információval találkozunk, kezdve a szoba nevével és jelszójával, hogy további játékosokat hívhassunk meg.

Látjuk a már belépett játékosokat, a profilképüket és egy indikátort arról, hogy készen vannak-e a játékra. Ehhez minden játékosnál megjelenik egy gomb, amivel ki-be tudják ezt kapcsolni. Ez az indikátor is élőben követhető, frissítés nélkül az oldalon.

A szoba tulajdonosa külön jogokkal rendelkezik. Először is módosítani tudja, hogy milyen paklival akarjuk játszani a játékot. Ez automatikus kiveszi a kész állapot indikátorát minden játékostól. Olyan paklit tud választani, amihez megvette a boltban a paklit. Emellett a többi játékos olyan pakli mellett tud kész gombot nyomni, aminek ő is megvette a pakliját. Tehát csak olyan paklival indulhat a játék, amihez mindenki megvette a hozzá illő paklit.

A másik joga a szoba tulajdonosának maga a játék elindítása. Ezt akkor teheti meg, ha minden játékos kész van, és szerepet játszik az is, hogy az adott paklinak mi az ajánlott alsó és felső határa a játékosszámának. Ha ez a két feltétel nem teljesül, akkor nem engedi elindítani a játékot a honlap.

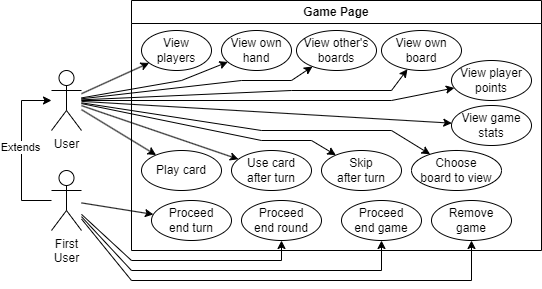
Egy további funkció a szobában, hogy a belépett játékosok tudnak élőben chatelni egymással, míg a játék indítására várakoznak. Ez időrendben megjelenik a szoba alján, ahol látják ki és mit írt a szobában.

### Játék rendszer

Az alkalmazás fő egysége maga a játéknak a megvalósítása. Erre a felületre egyedül a váróteremből elindított játék után lehet kerülni, és a játék befejezéséig vagy törléséig az oldalon bárhova lépve ide irányít át minket.

A játékosok egy átlátható felületet kapnak a játék állásáról. Jobb oldalt egy listán látják a játékosok listáját és sorrendjét, amin az elemekre kattintva a baloldali mezőn megjelenik az adott játékos asztalára kirakott lapok listája. Emellett a képernyő alján látják a saját kezükben található lapokat, amit átválthatnak az asztaluk nézetére, ha onnan akarnak indítani valamilyen akciót.

A játékban a körben első játékos felvesz egy vezető szerepet, neki külön jogai vannak. Minden játékos tudja, hogy kinek van az aktuális köre. Ennek a játékosnak az oldal jelzi, hogy hol és milyen akciókat tud végrehajtani. Így nem tud véletlen rossz lépést tenni. Egy átlagos játékmenetben egyszerű lapokat, vagy lapokkal végrehajtott akciókat kell végrehajtaniuk. További akciója van az első játékosnak, akinek a feladata elindítani az új kört és menetet, viszont ez is megtörténik a szerver által, ha egy idő után nem indítja el.



4. ábra Játék használati esetei

A játékosok látják a játék aktuális állapotát, kezdve a nevétől, aktuális menettől a játékfázis állapotáig. Emellett látják azt is, hogy ki hány pontot halmozott fel eddig a játékban. Amikor a játék véget ér, akkor egy listában megtekinthetik a végső helyezésüket és pontjukat. Ekkor az első játékosnak van joga törölni a játékot, ami mindenkit visszadob a főoldalra.

## Architektúra

### Áttekintő

Az elkészült alkalmazás több szempont szerint is egységekre bontható. Ezekben az egységekben többféle architektúra és tervezési minta valósult meg vagy lett felhasználva.

Mivel egy full-stack alkalmazás valósult meg, ezért elsősorban különválaszthatunk egy kliensoldali és egy szerveroldali komponenst. A kliensoldali része foglalkozik közvetlenül a felhasználói interakciókkal és a webes a környezet kezelésével. A szerveroldali komponens pedig az alkalmazás logikájának megvalósításával és az adatok perzisztens tárolásával és transzformációjával.

A kliensoldali komponens felépítése egyszerűbben néz ki, mivel egy egységes Angular webalkalmazás készült el. Ezen az alkalmazáson belül lettek feldarabolva az egyes felületek komponensei, és a hozzájuk tartozó szolgáltatások, guard-ok vagy WebSocket hubok.



5. ábra Architektúra

A szerveroldal viszont ennél bonyolultabb. A téma egyik fő eleme, hogy mikroszolgáltatásokra épülő alkalmazás kifejlesztése a cél. Tehát az alkalmazás felelősségek mentén konténerekre lett osztva. Ezeken a konténereken belül egy-egy háromrétegű architektúrát megvalósító szerveralkalmazás található. Felülről számolva egy API réteg, egy üzleti logikai réteg és egy adatelérési réteg. Ezeken az alkalmazásokon belül további ismert tervezési minták lettek megvalósítva, mint például a CQRS, repository vagy unitofwork.

### Adatbázis

### Szerveroldal

### Kliensoldal

# Önálló munka bemutatása

## Bevezetés

## Szerveroldali funkciók

## Kliensoldali funkciók

## Felhasználókezelés

## Bolt kialakítása

## Váróterem megvalósítása

## Játék

# Összefoglaló

# Irodalomjegyzék

1. Docker alapozó [Online]:  
   <https://thebojda.medium.com/docker-alapoz%C3%B3-b8efb6aa68e9>
2. ASP.NET CORE dokumentáció [Online]:  
   <https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-7.0>
3. Ocelot dokumentáció [Online]:  
   <https://ocelot.readthedocs.io/en/latest/index.html>
4. MediatR wiki [Online]:  
   <https://github.com/jbogard/MediatR/wiki>
5. SignalR dokumentáció [Online]:  
   <https://learn.microsoft.com/hu-hu/aspnet/core/signalr/introduction?view=aspnetcore-7.0&WT.mc_id=dotnet-35129-website>
6. Redis dokumentáció [Online]:  
   <https://redis.io/docs/>
7. RabbitMQ dokumentáció [Online]:  
   <https://www.rabbitmq.com/documentation.html>
8. IdentityServer4 dokumentáció [Online]:  
   <https://identityserver4.readthedocs.io/en/latest/>
9. Hangfire dokumentáció [Online]:  
   <https://docs.hangfire.io/en/latest/>
10. Angular Material dokumentáció [Online]:  
    <https://material.angular.io/>
11. NGX-Translate dokumentáció [Online]:  
    <https://github.com/ngx-translate/core>

# Függelék