# Cross platform kártyajáték fejlesztése Spring és Flutter környezetben Dokumnetáció

# Specifikáció

## A projekt ismertetése

Az alkalmazásunk egy közismert kártyajáték, a Shed / Holland kocsma mobilos és webes megvalósítása Java Spring és Flutter környezetben. A játékot francia kártyával játszhatja 2-10 ember, 1, illetve 2 paklival is, igény esetén Jokereket is felhasználva.

## A részletes szabályok a következők

Minden játékosnak kiosztunk 3-3-N kártyát, amik rendre az asztalon lefordítva, az asztalon felfordítva és a kézben lesznek. Az N lehet 3, 4, illetve 5 is. A kártyákat sorban rakják a játékosok egymás után és ha a kártya szabály mást nem mond, növekvő sorrendben kell rakni. A játék célja a kártyáinktól minél hamarabb megszabadulni.

Bármelyik kártyának beállíthatunk egy adott szabályt, az alábbiak közül:

**Jolly Joker** - Ezt a kártyát bármire tehetjük és erre a kártyára bármit tehetünk

**Kicsinyítő** - Erre a kártyára csak kisebb kártyát szabad rakni

**Átlátszó** - Ezt bármikor lerakhatjuk, rá viszont úgy kell rakni hogy az alatta lévő lapot vesszük figyelembe

**Égető** - Ezt csak az alapszabály szerint rakhatjuk (nála kisebbre, vagy az alatta lévő kártya szabályainak megfelelően) és kitörli a játékból a rakó lapokat magával együtt

Ha a rakott lapok legfelső 4 lapjának száma megegyezik, akkor a pakli szintén ‘ég’, azaz kitörlődik a játékból az összes rakó lap, többet már nem lehet őket felhúzni.

Amennyiben egy játékos nem tud rakni szabályosan, fel kell hogy húzza az összes lapot.

Ha valaki kedve szerint akar húzni egy lapot, de amúgy tudna rakni szabályosan, akkor a húzó lapokból is tud egyet felvenni a kezébe.

Ha a húzó lapok elfogytak, akkor a játékosoknak először a kezükben lévő lapoktól kell megszabadulniuk. Amint megvannak ezzel, az asztalon lévő felfordított lapokból kell rakniuk, de ez csak akkor megengedett, ha azt szabályosan teszik meg (olyat nem szabad, hogy szabálytalanul raknak aztán felhúzzák a kezükbe a felfordított lappal együtt a paklit)

Ha a felfordított lapok is elfogynak, akkor véletlenszerűen kell felcsapniuk a lefordított lapok valamelyikét, amit utána mindenki más lát. Ha a lap rakása nem szabályos, akkor kénytelenek felhúzni az összes lapot a rakó pakliból, a felcsapottal együtt is, ezek a kártyák a kezünkbe kerülnek.

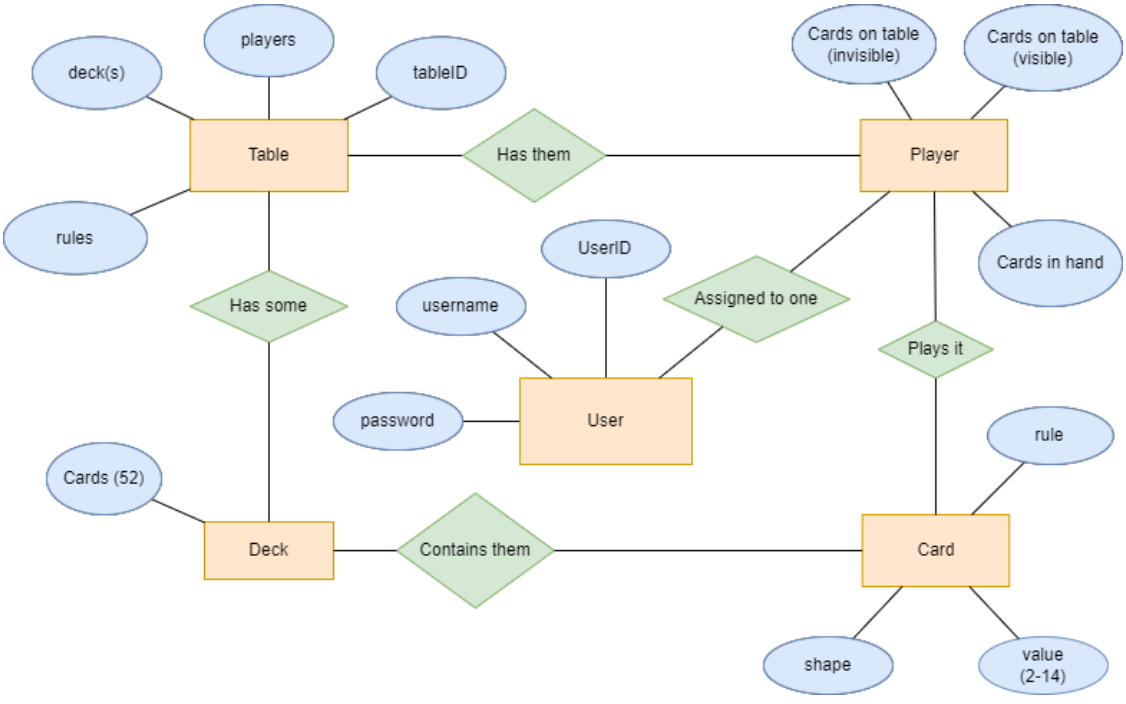
Ha egy kivételével az összes játékos lapja elfogyott, akkor ki is hirdethetjük a vesztest, aki értelemszerűen az utolsó játékos lesz.

# Haladási napló

## 1. hét

Mivel a konzin felmerült hogy websocketet vagy mqtt protokollt használjunk a kommunikációra, ezért erről beszéltünk és végül az mqtt mellett döntöttünk. Azért emellett döntöttünk, mivel azt olvastuk róla, hogy - Hatékonyan osztja szét az információt a kliensek között, ami azért jó, mert nem a felhasználóknak kell szűrniük az információt - Minimális sávszélességet igényel, ez pedig esetünkben elég fontos tényező. - Lightweight és gyorsan összerakható, ezzel sok fejelsztési időt lehet spórolni - Megbízható, skálázható és bármilyen adatot lehet rajta közvetíteni

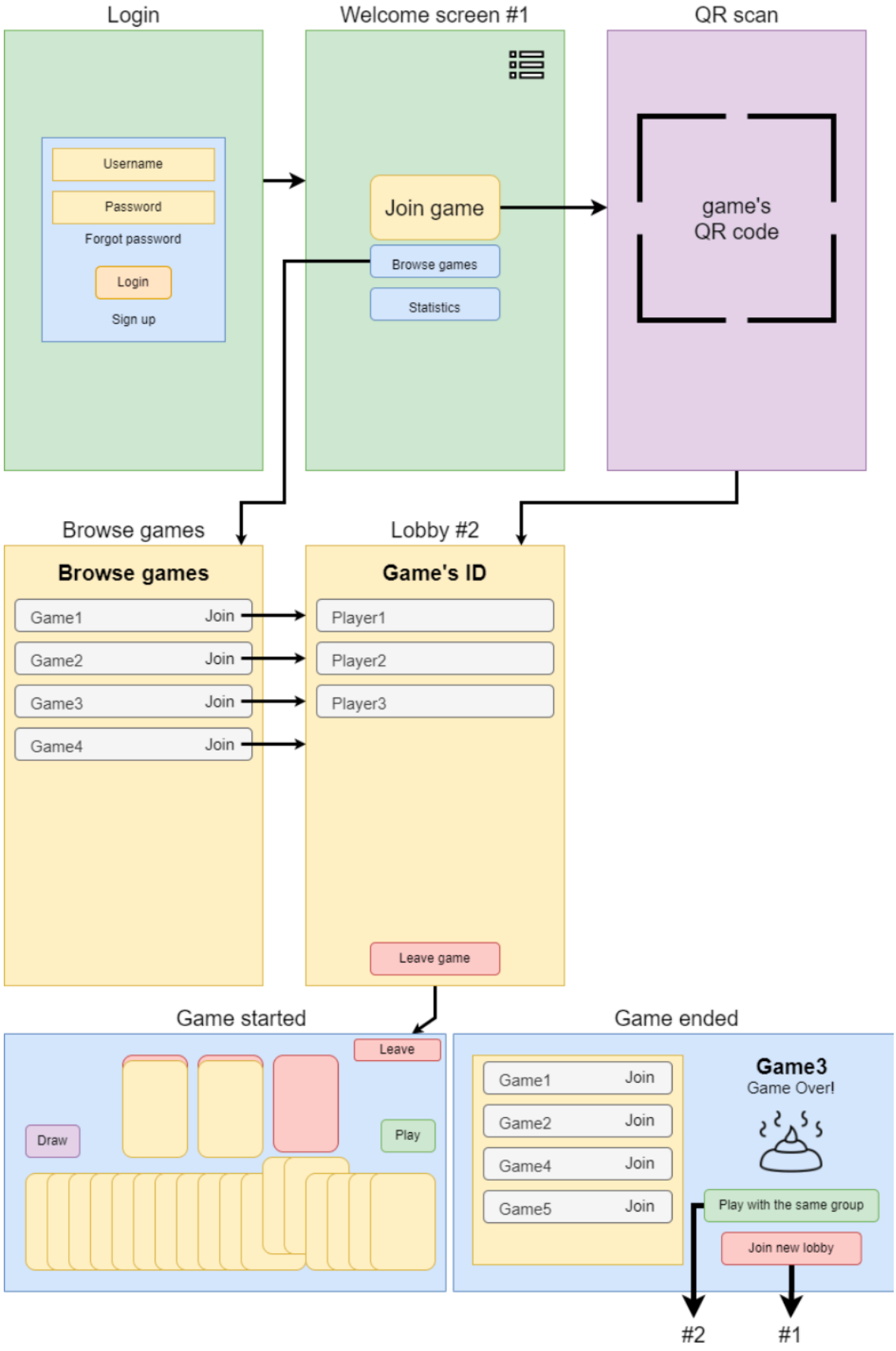
Elkészítettük a projekt specifikációját és megterveztük az adatmodellt, ami itt látható:




*adatmodell*

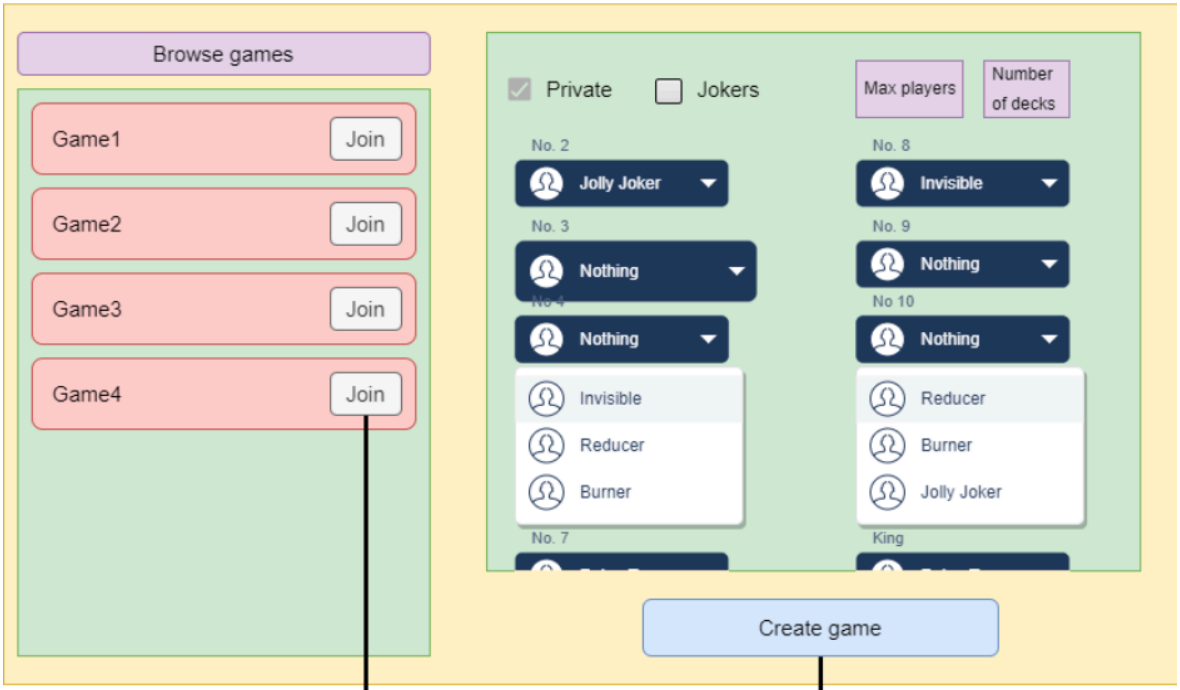
Ezek után megbeszéltük, hogy hogyan fogjuk szétosztani a munkát az első pár héten. A terv az, hogy amíg nincsen kész az alap kommunikációt biztosító backend, addig közösen dolgozunk rajta, majd ha elkészültünk vele és képesek leszünk bármiféle információt átadni rajta, elkezdjük a mobil kliens fejlesztését is. Előre láthatólag Tomi fog többet foglalkozni a backenddel, én pedig a Flutteres résszel, viszont mindketten fogunk mindkét oldallal

# Flutter mobile app

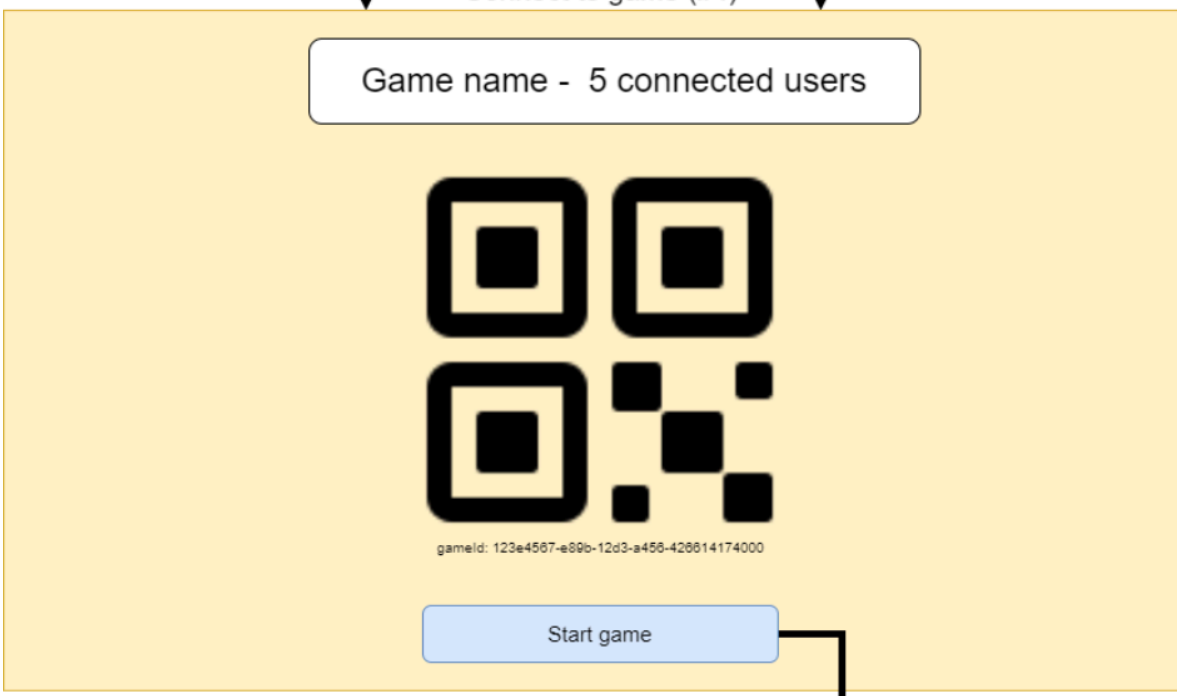


# Flutter Web app

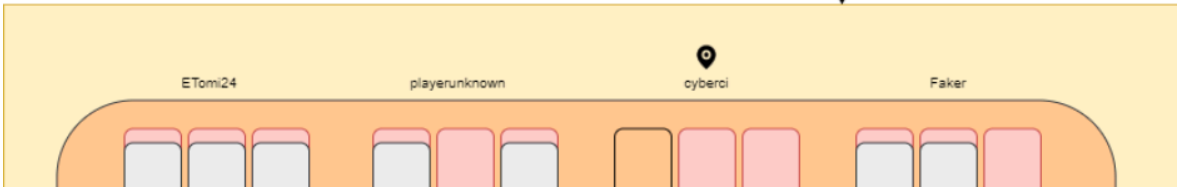
## Web app welcome screen (#3)



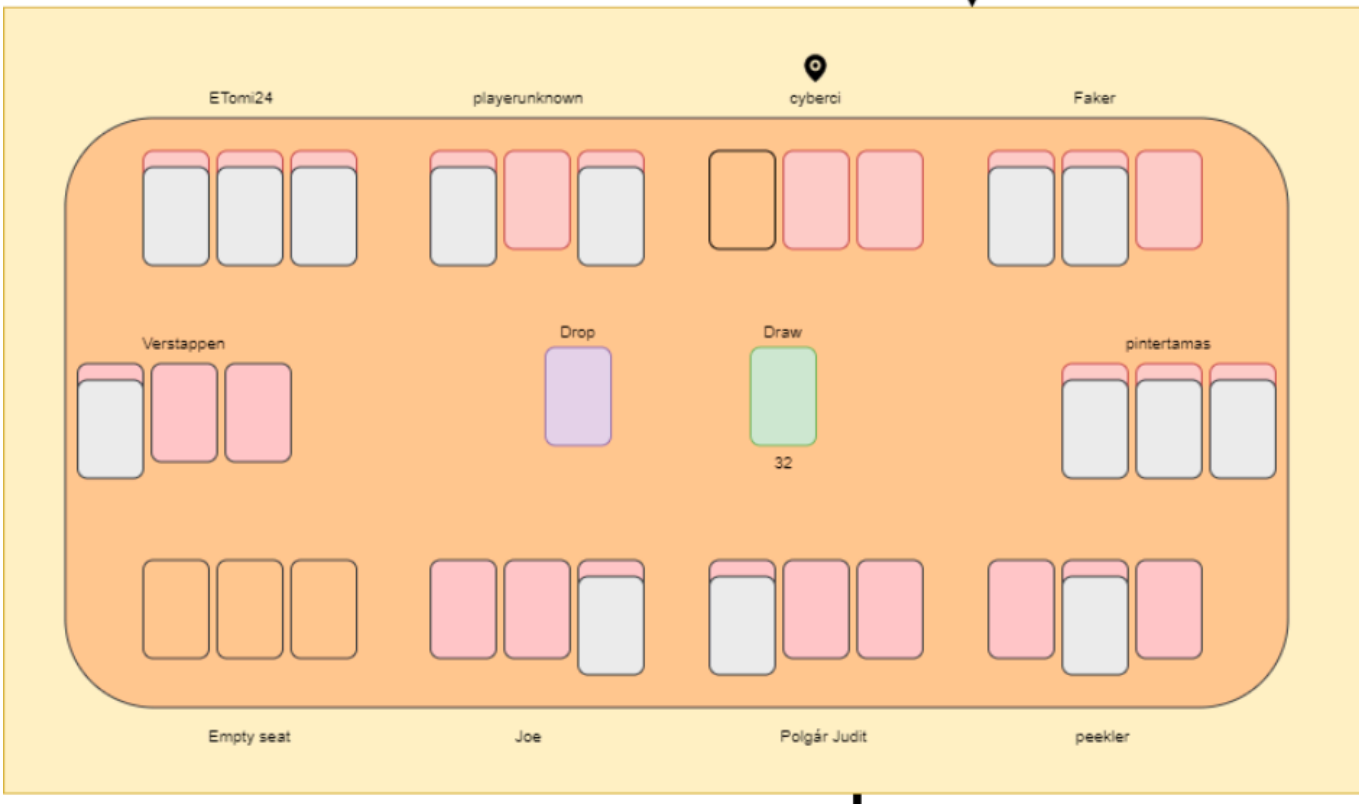

## Connect to game (#4)




## Game in progress



Game in progress




Game Over

# Game Over!

## Worst 3

- Player1
- Player2
- Player3

New game

Leave game

#4

#3

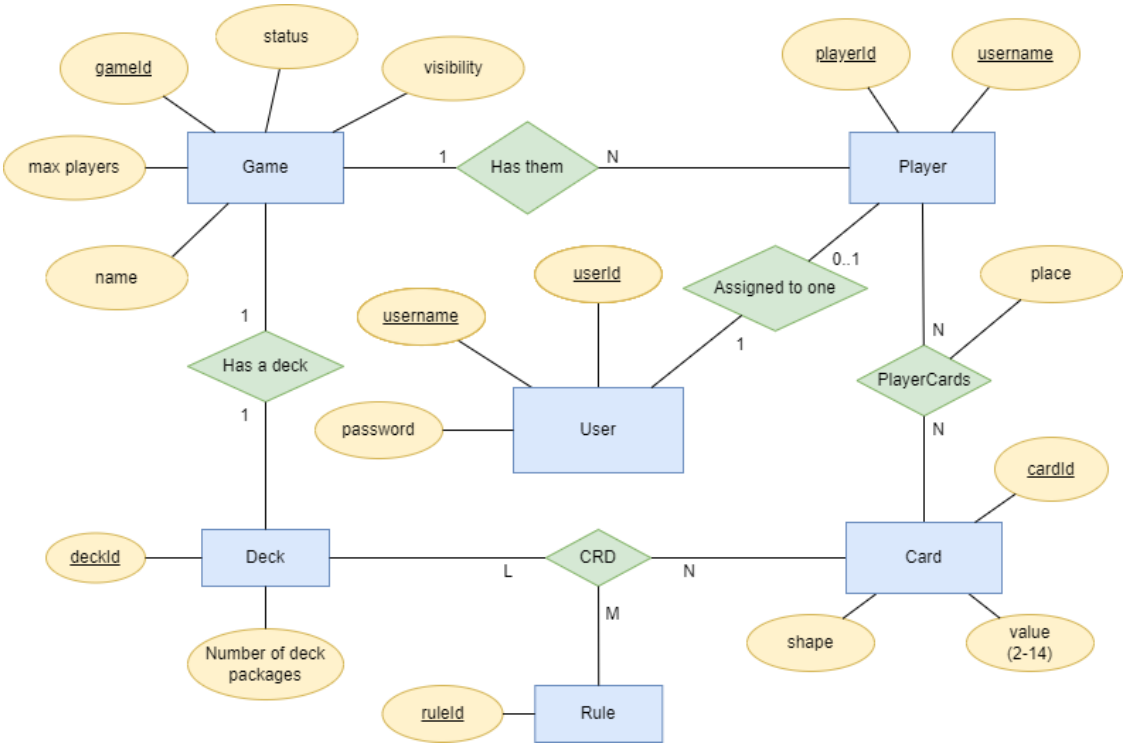
foglalkozni, a feladatokat pedig taskboard használatával fogjuk elosztani.

##2. hét A levelezős konzultáció után megpróbáltunk egy rendes ER diagramot készíteni és frissítettük a specifikációt, így már látható a játék részletes szabálya és el is képzelhető hogy a felhasználók mit fognak látni mikor játszanak a játékkal.

Kicsit jobban utánanéztünk hogy hogyan tudnánk a gyakorlatban használni a RabbitMQ-t és arra a döntésre jutottunk hogy a websocketes megközelítés valószínűleg találóbb lesz a kétirányú kommunikáció miatt. Megcsináltunk egy Spring alapú websocketes kommunikációt lehetővé tevő alkalmazást és Flutterben is csináltunk hozzá egy demo alkalmazást amivel websocketen lehet küldeni és fogadni üzeneteket. Csináltunk egy SSL Certificatet az alábbi tutorial segítségével, így már HTTPS-t használ az alkalmazásunk.

## 3. & 4. hét

A 3. hét elején újra terveztük az adatmodellünket, viszont a két hét alatt nagyon sokat változott. Először erre változtattuk:




Ezzel meg is voltunk elégedve, de miután nekiálltunk megvalósítani, azt beszéltük meg hogy frissítjük, úgy, hogy az adatbázis táblák így nézzenek ki:

CardConfig

| Card no. | shape | rule | gameId |
|---|---|---|---|
| 1 | 3 | 2 | 2 |
| 2 | 4 | 1 | 1 |

PlayerCards

| cardConfigId | playerId | state |
|---|---|---|
| 1 | 5 | 0 |
| 3 | 4 | 1 |

TableCards

| cardConfigId | state | |
|---|---|---|
| 2 | 1 | |
| 4 | 0 | |

User

| userId | username | password |
|---|---|---|
| 5 | Tomi | ***** |
| 4 | Tomi2 | ***** |

Game

| gameId | name | cardsInHand | jokers | status | visibility |
|---|---|---|---|---|---|
| 1 | game1 | 3 | 0 | 0 | 1 |
| 2 | game2 | 4 | 1 | 1 | 0 |

Player

| playerId | gameId | username |
|---|---|---|
| 1 | 1 | Tomi |
| 2 | 1 | Tomi2 |

A tervünk az, hogy a felső 3 táblát Redis-t használja cache memóriában tároljuk mivel ezeket folyamatosan olvasni és írni kell, az alsó 3-at pedig PostgreSQL adatbázisban (mivel ezeket az adatokat ritkán változtatjuk, de hosszú távon kell tárolni), amit azóta már setupoltunk is.

####RedisDB: - A CardConfig tábla azt fogja tárolni, hogy adott kártya szám - szín kombinációhoz melyik szabályt rendeltük adott játékban. - A PlayerCards arra lesz jó, hogy a játékosok kezeiben lévő kártyákat számon tudjuk tartani olyan módon, hogy a játékos id-ja alapján lekérdezhetjük a kártyái beállításait (CardConfig) és a kártya állapotát (asztalon lefordítva/felfordítva, vagy a kezében) - A Table cards ugyan erre van, csak az asztalon lévő kártyákat tárolja és azt, hogy húzó vagy rakó pakli rész-e

####PostgreSQL: - A User tábla egy felhasználó adatait tárolja (felhasználónév, jelszó, valamint a felhasználó id-ját) - A Game táblába a játékok elején írunk, mikor létrehozunk egy játékot és a státuszának változtatásakor szerkesztjük - A Player tábla a játékosokat reprezentálja, segítségével le lehet kérdezni adott játékban lévő felhasználók neveit, ami jól fog jönni az asztalnál ülő játékosok neveinek kiírásához.

Hogy jobban elképzelhető legyen a játék, csináltam egy layout tervet draw.io-val, amit itt lehet megnézni:

A backenden elkezdtük megírni az alapokat, már tudunk regisztálni, bejelentkezni, meg a játék létrehozás is működik valamilyen szinten. Hostoltuk Herokura is, amin hiba nélkül futott, lehetett becsatlakozni játék lobbykba és üzeneteket küldeni egymásnak. A Flutteres részen létrehoztam egy skálázható mappa architektúrát ennek a cikknek alapján. Mobilon pedig be lehet járni az alkalmazás nagy részét, bescannelni qr kódokat és a bennük tárolt játék nevek alapján játékokba becsatlakozni. Weben még nincs sok minden meg, de lehet generálni QR kódot ami kódol egy szöveget, ezt a kódot pedig megjeleníti a képernyőn, amit be lehet scannelni mobillal. A login tesztelésekor belefutottam egy hibába, mindig timeoutolt a post kérésem. Azt a feltételezhető okot találtam erre, hogy a gépem egyik portjára küldöm a hívást és ez okozza a bajt. Ekkor kiraktam Herokura remélve hogy így már jól fog működni, viszont sajnos teljesen eltörtem az appot és most azt az üzenetet kapom a mobilon hogy `WebSocketException: Connection to 'https://shed-backend.herokuapp.com:0/shed/018/1wnesojm/websocket#' was not upgraded to websocket` Ennek a kijavítását a jövő hétre hagyom, remélhetőleg könnyen megjavul, mivel már futottam bele ebbe a hibába korábban és valahogy megoldottam.

## 5. hét

Ezen a héten megfejtettük, hogy a websocket eltörését mi okozza. Ebben nagy segítséget nyújtott az, hogy Herokun megtaláltuk a logok helyét, ahol azt láttuk, hogy a "/login" endpoint elérésekor a JWT tokenre panaszkodik a szerver, aminek annál a fázisnál még nem kellene szóba jönnie. Rájöttünk hogy a regisztráció rosszul lett megírva, úgyhogy azt átírtuk és így már működött a login. Ekkor visszakaptuk a megfelelő Bearer tokent, amit így már be tudtunk adni a websocket fejlécébe, aminek hatására újra rendeltetésszerűen működött a szerver. Elkészítettem a mobil appon egy töltő képernyőt, ami megkérdezi a szervertől, hogy a bejelentkezéskor shared_preferences-be lementett token érvényes-e még és ettől függően tovább irányít 1) egy login/register opciók közül választást kínáló oldalra, 2) az "üdvözlő" képernyőre, ahol választhatunk hogy mit akarunk a továbbiakban csinálni bejelentkezett felhasználóként.

## 6. hét

Elkezdtem haladni a webes résszel is, aminél egy problémába akardam már az elején. Nem volt konfigurálva a CORS a backenden, ezért webről nem tudtam hívásokat küldeni a backend felé. Ahhoz, hogy ez megoldódjon, követtem a leírást a következő oldalon: `https://spring.io/guides/gs/rest-service-cors/` A globális beállítást választottam és így már működött az API hívás, meg tudtam jeleníteni egy új játék nevét és az ahhoz tartozó QR kódot a képernyőn.

## 7. hét

Megcsináltam, hogy a becsatlakozott játékosok lekérik az addig belépett playerek listáját, onnantól kezdve meg websocketen keresztül érkező join illetve leave üzenetek hatására frissítik a lobby megjelenését. game-start üzenet hatására átnavigálnak a játék képernyőjére.

## 8. hét

Ezen a héten megcsináltam egy one time password rendszert az alkalmazáshoz, ami azt teszi lehetővé, hogy a regisztráló játékosok csak az emailben megkapott kód beírása után lesznek csak ténylegesen elmentve. Ezek a jelszók a szerveren vannak egy cache tárolóban, ami úgy működik, hogy általam megadott ideig (5 perc) vannak tárolva, aztán kitörlődnek automatikusan, így a felhasználónak ennyi ideig van lehetősége felhasználni azt saját maga azonosításához. Ehhez még szépítettem is az email küldő funkcióban lévő email templatet, hogy barátságosabban nézzenek ki az emailek. Mobilon megcsináltam a bekért inputok validációját, ehhez létrehoztam egy külön service-t, valamint a hibaüzeneteket mutató popupot is elkészítettemm így lesz visszajelzés a usereknek arról, hogy miért akadnak el pl. a regisztrációban.

## 9 & 10. hét

A mobilos regisztrációkhoz tartozó one time password képernyőt megcsináltam dizájnosra és beraktam egy jelszó újraküldésre alkalmas gombot, amit egy percenként tudnak újra használni a felhasználók, ezzel levéve a terhelést a szerverről. Megcsináltam weben a játékok böngészését, ami úgy működik, hogy pár másodperces rendszerességgel lekéri a kliens az új játékokat a szervertől és frissíti a listát. A mellette lévő szabály beállíthatóságot is megcsináltam, így mostmár olyan játékot tudunk létrehozni amilyet csak akarunk sliderek, switchek meg dropdown buttonok segítségével. A QR képernyő készítése közben eldöntöttem hogy kezdek valamit azzal a problémával, hogy a hot reloadnál teljesen újratöltődik az alkalmazás és még a legkisebb változtatások megnéséséért is újra létre kell hoznom egy játékot, ami sok időt vesz igénybe. Főleg azért akartam minél hamarabb megoldani ezt, mert a játék képernyőn dolgozva egyre nehezebb lesz ilyen formájában tesztelni a kinézetbeli változtatásokat. Nem igazán találtam erre megoldást az interneten, de egy megoldás volt, aminél egyrész egértettem hogy a hiba az, hogy a routingomban pont a QR képernyőnél argumentumokat passzolok át a képernyőnek, ami miatt nem a sima routingot használtam, hanem az ongenerateroute-ot, viszont nem állítottam be a settings-t az új képernyőnél, ezért az összes oldalt a create game URI-val jelenítette meg. `https://stackoverflow.com/questions/62442045/page-url-not-changing-in-flutter-but-the-page-content-changes-fine` Ezt megjavítva most van egy workaround megoldásom a problémára, ami úgy néz ki, hogy ha hot reloadolok akkor ott marad a

képernyő, csak a képernyő argumentek nullázódnak, ami miatt le kell kezelnem ezeket az értékeket. Egyelőre a játék képernyőnél nem tudom hogy fogom ezt megoldani mert feltehetően csomó adat lesz ott amiket nem akarnék placeholderekkel helyettesíteni. Ezek után megcsináltam a QR képernyőn a játékba becsatlakozó játékosok listázását, így most a képernyőn is látszik ugyan az a lista ami telefonon is.

## 11. hét

Találtunk egy nagy hibát a websocketünkkel, amit nagyon sokáig debugoltunk, mert nem találtuk meg hogy mi okozhatja. A probléma az volt, hogy amikor bekerült a webes képernyőkre is a lobby, akkor a frontendre rengeteg stacktrace hibaüzenet jött stompBadStateException néven. Úgy oldottuk ezt meg, hogy én nyomoztam a kód websocketes részeiben, Tomi pedig a Herokus logokat mondta nekem. Egy idő után (és plusz logok bevezetésével) feltűnt, hogy gyanúsan sokszor próbál a webes kliens becsatlakozni a websocketre és ezt a tudást felhasználva megtaláltuk frontenden a hibát, ami az volt, hogy a Lobby képernyőt hasznosítottam újra weben is, viszont mivel mobilon ez a képernyő csatlakoztatja be a játékosokat a websocketre, ez felülírta a weben már meglévő websocket kapcsolatot. Ezzel nagyon sok időnk ment el, de szerencsére most már minden jól működik. Mobilon kijavítottam a register és login képernyőn egy overflow bugot és bevezettem projekt szinten a gombok letiltását, így most kattintás után nem működnek amíg el nem végzik a feladatukat. Megcsináltam a mobilos játék képernyőt, amin már lehet kártyákat kiválasztani (még nem teljesen tökéletes, javításra szorul), valamint a websocketre üzeneteket küldeni. A terv az, hogy a backendnek küld a mobil egy olyan üzenetet, ami a kártyák listáját tárolja id-val együtt és egy usernevet, amit amikor megkap a backend, kiértékel (hogy szabályos-e a lépés) és válaszban visszaküldi a sikerességet, hiba esetén meg a hiba megnevezését. Abban az esetben ha a játékosnak kevesebb, mint 3 lapja van, vagy húznia kell midnenképpen, a szerver a válaszban elküldi a húzott lapokat is. Ha a mobil azt kapja, hogy sikeres volt a lépés, akkor kitörli a lerakott kártyáit, ha viszont sikertelen volt, akkor kitörli a választott kártyák listáját. A kártyák állapotához a sima Provider könyvtárat használom, egyelőre még vannak vele hibák, de majd ha teljseen megértem a működését akkor szerintem elég lesz erre a feladatra. A kártyák animálására az a tervem, hogy a választott kártyák megnőnek és a lerakásukkor egy oldalra csúszó animációt fogok használni. Mivel nem vagyok annyira tisztában az animációkkal, ezért kerestem egy könyvtárat ami ilyenekre képes és az AnimatedList-et találtam, ami segítségével egész szép animációkat fogok tudni készíteni, remélhetőleg kevés energia befektetéssel is. A jelenlegi célom a mobilos képernyő tökéletesítése, főképp a websocketes kommunikációé. Ezek után fogom majd a webes képernyőt megcsinálni.

## 12. hét

A kártya rakás hibáin dolgoztam, hogy rendesen lehessen rakni kártyát a játékosoknak, ami után a képernyő frissül és az újonnan húzott kártyák is látszódnak. Egyelőre nem működik a képernyő frissítés és a húzott kártyák parsolása, de a websocketen való rakó kártyák küldése működik és a visszajövő üzeneteket is le tudja már kezelni az alkalmazás.

## 13. hét

Az előző heti problémát sikerült végre megoldanom. Arra a problémára, hogy a provider ne a StreamBuilder return-je előtt, a snapshot kiértékelése közben változtassa az adatokat ezzel túl korán (a notifyListeners függvény miatt) setState-et hívva, beleraktam a kód blokkot egy WidgetsBinding callbackbe a következő módon:

```
WidgetsBinding.instance?.addPostFrameCallback(
    (_) => {
        game.deletePlayedCards(),
        getGameCards.then((value) => {
            game.setCards(value),
        },
    ),
},
```

Így most a kód csak az után fut le, hogy a widgetek betöltöttek ezzel elkerülve a korai újrabuildelésüket. Szintén a game_screen.dart fájlban változtattam a loadData függvényen, így minden provider változtatáskor szinkronizálom a szerveren lévő adatokat a mobillal amiről egy FutureBuilder gondoskodik. Egy probléma azomban felmerült, ami az, hogy egészen lassan töltenek be a kártyák a lap rakás után, ami kicsit rontja a felhasználói élményt. Ez valószínűleg a WidgetsBinding-nak tudható be, de egyelőre nem tudtam megtalálni az okát. Sajnos belefutottunk egy nagyobb problémába is, amiből sokat tanultunk. A szabályokat nem beszéltük meg elég részletességgel és emiatt lettek félreértések a kódban, amiket nehéz volt megjavítani, néhányat el is vetettünk. Azt tanultuk meg ebből, hogy a specifikációnak elegendően részletesnek kell lennie, ahol meg homályos valami mégis, ott meg pontosítást kell kérni róla.

Képek a megvalósult appról: