



Eötvös Loránd Tudományegyetem
Informatikai Kar
Információs Rendszerek Tanszék

Többasztalos és -felhasználós póker játék adatbázis modellezése

Ács Zoltán
tanársegéd

Fehér Valentin
Programtervező Informatikus BSc

Budapest, 2016

Tartalomjegyzék

	Oldal
I. Bevezetés	3
1. A téma választás indoklása	3
2. Feladat leírás	3
II. Felhasználói dokumentáció	4
3. Követelmények	4
3.1. Hardver	4
3.2. Szoftver	4
4. Telepítés	4
4.1. Java SE Runtime Environment 8	4
4.2. MySQL Community Server 5.6	5
4.3. Az adatbázis használatba vétele	6
5. Fogalomtár	7
5.1. Parancsok	7
5.2. Fogalmak	8
6. A póker játékról	8
6.1. Játékmenet	9
6.2. Játékstílusok	10
7. Futtatás	11
7.1. A póker szerver elindítása	11
7.2. A póker kliens elindítása	11
8. A póker játék használata	12
III. Fejlesztői dokumentáció	16
9. Felhasznált technológiák	16
9.1. Eclipse	16
9.2. Maven	16
9.3. MySQL	17

9.4. Git	17
9.5. Enterprise Architect	18
10. Adatbázis séma	18
10.1. Póker játék asztal tábla	19
10.2. Felhasználók tábla	19
10.3. Játékstílus tábla	20
11. Megoldási terv	20
11.1. Probléma	20
11.2. Tervezés	21
11.3. Implementálás	24
11.4. Elemzés	26
12. Modulok	27
12.1. Model modul	27
12.2. Shared modul	29
12.3. Persist modul	30
12.4. Kliens modul	31
12.5. Szerver modul	33
12.6. javapokertexasholdem modul	34
13. Funkciók	34
13.1. Általános funkciók	34
13.2. Játékmenetbeli funkciók	37
14. Tesztelés	38
14.1. Funkcionális tesztelés	38
15. Tovább fejlesztési lehetőségek	39
IV. Irodalomjegyzék	41
16. Hivatkozások	41

I. rész

Bevezetés

1. A téma választás indoklása

Mindenképpen egy online, többfelhasználós játékot szerettem volna megvalósítani. Később leszűkítettem a kört kártyajátékokra, és végül az ulti és a póker között vaciláltam. A döntésem a pókerre esett, ugyanis az ulti viszonylag bonyolultabb, mint a póker, több szabály, több megszorítás a partykra vonatkozólag, ráadásul ahány ház annyi szokás alapon könnyen nézet eltérések szoktak keletkezni az ulti-zás során. Pókerezni egyszerűbb - bár ezt sokan vitatják - és jó formán mindenki könnyen megérte a játék lényegét.

2. Feladat leírás

II. rész

Felhasználói dokumentáció

A programcsomagot JAVA programozási nyelven írtam meg, így a program számára biztosítani kell JAVA futtatókörnyezetet, továbbá a program működéséhez adatbázisszerverre is szükség lesz. A telepítés magába foglalja az adatbázisséma elkészítését és az adatbázis demóadatokkal való feltöltését is, így a telepítést követően extra dolgokra nem lesz szükség a szoftver használatához. A telepítés közben a számítógépet újraindítani nem szükséges.

3. Követelmények

3.1. Hardver

- Legalább Intel® Pentium® 4 Processor 2.80 GHz
- Legalább 1GB RAM
- Legalább 50MB szabad lemezterület

3.2. Szoftver

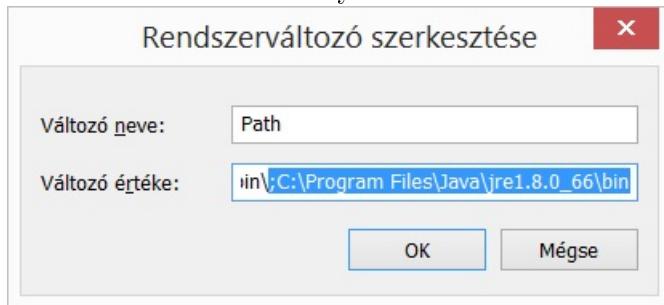
- Legalább Windows XP operációs rendszer
- Java SE Runtime Environment 8
- MySQL Community Server 5.6

4. Telepítés

4.1. Java SE Runtime Environment 8

A programcsomag futtatásához legalább Windows XP operációs rendszer szükséges, amelyen Java SE Runtime Environment 8 futtató környezet [1] (a továbbiakban: JRE) fut. A JRE feltelepítését követően manuálisan ellenőrizzük, hogy a rendszer felvette-e környezeti változóként az installációs könyvtárat. Navigálunk az operációs rendszerben a környezeti változók módosítása panelhez, majd ellenőrizzük le, hogy a PATH nevű környezeti változóhoz hozzá lett-e adva az installációs könyvtár, mint például: C:\ProgramFiles\Java\jre1.8.0_66\bin. A telepítési könyvtár operációs rendszerenként eltérhet.

1. ábra. PATH környezeti változó értéke



Ha a JRE installációs könyvtár gyökerében található bin könyvtár nincs behivatkozva a PATH nevű környezeti változó értékéhez, akkor az 1. ábrán látható módon egészítsük ki az értéket, majd mentsük el a változtatásokat és indítsuk újra a promptot. Ha sikeresen jártunk el, akkor a

```
java -version
```

utasítás hatására a 2. ábrán látható szöveg jelenik meg a konzolon, akkor sikeres volt a JRE telepítése és beállítása.

2. ábra. JRE verzió

```
D:\projects\szakdolgozat\poker>java -version
java version "1.8.0_66"
Java(TM) SE Runtime Environment (build 1.8.0_66-b18)
Java HotSpot(TM) 64-Bit Server VM (build 25.66-b18, mixed mode)
```

4.2. MySQL Community Server 5.6

A programcsomag megköveteli a MySQL Community Server 5.6 [2] adatbáziskezelő rendszer (a továbbiakban: MySQL Server) használatát is. Letöltés után cso-magoljuk ki a zip állományt egy tetszőleges könyvtárba, majd az előbekkel meg-egyező módon adjuk hozzá a PATH nevű környezeti változó értékéhez a MySQL Server bin könyvtár elérési útvonalát. Ha a prompt nyitva van, akkor zárjuk be és indítsuk el újra, majd navigálunk a MySQL Server bin könyvtárába, ott pedig adjuk ki a

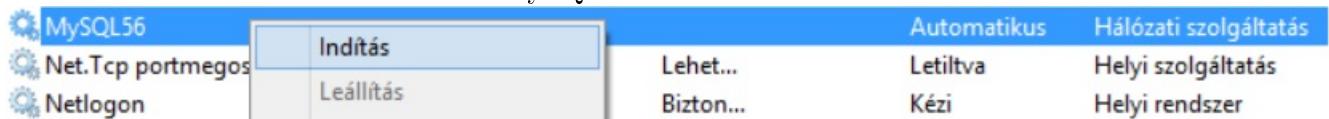
```
mysqld --install
```

parancsot. A parancs végrehajtása után navigálunk a szolgáltatások panelhez, amelyet legkönnyebben a promptban a

```
services.msc
```

kiadott utasítással lehet elérni. Majd járunk el a 3. ábrának megfelelően.

3. ábra. MySQL Service



Térjünk vissza a konzolra, ahol adjuk ki a

```
mysql -u root -p
```

parancsot, amely jelszót fog kérni. A beviteli sort hagyjuk üresen, nyomjunk enter-t.
Ha sikeresen beléptünk az adatbázis-kezelő rendszerbe, akkor adjuk ki a

```
SELECT VERSION();
```

utasítást, és ha a 4. ábrának megfelelő képernyőképet kapunk, akkor sikeresen feltelepítettük az adatbázis-kezelő rendszert.

4. ábra. MySQL Server verzió

```
mysql> SELECT VERSION();
+-----+
| VERSION() |
+-----+
| 5.6.27 log |
+-----+
1 row in set (0.00 sec)
```

4.3. Az adatbázis használatba vétele

Ha sikeresen elindítottuk a MySQL Server-t, akkor szükségünk lesz egy új adatbázisséma-re és demóadatokra, amelyet a X:\poker\release\poker-db.sql állományban találunk. Az X kötetcímke jelöli jelenleg és a későbbiek során is a DVD meghajtót, amelybe a DVD lemezt helyeztük. Ezt a fájlt kell lefuttatni az adatbázis szerveren. Egymás után többszöri futtatás esetén is ugyanazt az eredményt kapjuk.
Az adatbázis-kezelő rendszerből lépjünk ki a

```
quit
```

parancsal, majd a promptban adjuk ki a

```
mysql -u root -p < X:\poker\release\poker-db.sql
```

utasítást. A parancs kiadását követően az adatbázis-kezelő rendszer jelszót fog kérni.
A beviteli sort ugyancsak hagyjuk üresen. Ha sikeresen lefutott a parancs, akkor az adatbázisséma elkészült és a program demóadatokkal is feltöltötte.

5. Fogalomtár

Itt helyezkedik el minden olyan fogalom, amely a játék használatához feltétlenül szükséges. A dokumentáció későbbi fejezetei ezekre a fogalmakra támaszkodnak.

5.1. Parancsok

Az alábbi parancsokat a felhasználók tudják kiadni. A szerver biztosítja, hogy csak a megfelelő parancsok legyenek hívhatóak a póker partik során.

CALL

A felhasználó ezt a parancsot adja ki, ha a tartozását ki szeretné egyenlíteni az asztal felé.

CHECK

Játékos által kiadható parancs, melyet akkor kell kiadnia, amikor a játékos nem tartozik az asztal felé, és emelni sem szeretné a tételet.

BLIND

Olyan parancs, amelyet a kliens oldali program automatikusan hív meg. Ha a szerver BLIND típusú utasítást küld, akkor a kliens automatikusan beteszi a kis vagy nagy vakot az asztalra attól függően, hogy a szerver mire kötelezte az adott felhasználót. A kis vak, vagy angol nevén small blind az asztalnál érvényben lévő alaptét felét, míg a nagy vak, vagy angolul big blind az asztalra vonatkozó alaptét egészét jelenti.

RAISE

Olyan speciális parancs, amely hatására a felhasználó tartozása kiegyenlítésre kerül az asztal felé és még emel a téten az asztal alaptétjének megfelelő értékkel.

QUIT

Asztal elhagyására szolgáló parancs. A kilépését követően a program visszairányítja a felhasználót a táblalistázó oldalra. Ha bármilyen oknál fogva a gomb letiltásra kerül és a játékos nem tudja elhagyni az asztalt, ez esetben a billentyűzeten a Q gombot megnyomva visszakerül a játékos a táblalistázó felületre. Fontos megjegyezni, hogy a gomb a póker partik közben csak akkor van engedélyezve, ha éppen az adott felhasználó következik. Ha az asztal megfelelően működik és a játékos mégis úgy dönt, hogy ezt a funkciót kihasználva hagyja el a játékasztalt, akkor a gondolkodási idő lejárta után az asztal eltávolítja a játékost az asztaltól és a parti folytatódik tovább. A kliensnél ekkor a táblalistázó felület töltődik be.

CHANGE

A parancs csak a classic játékstílusban érhető el. A parancs meghívásával a felhasználó utasítja a programot, hogy a cserére megjelölt kártyalapokat a szerver új kártyalapokkal helyettesítse. Az új kártyalapok csak akkor kerülnek kiosztásra, ha minden játékos nyilatkozott az adott körben.

LOG

A felhasználó játék közben korlátozott mértékben vissza tudja nézni a már megtörtént főbb eseményeket. Például az előző játékos milyen parancsot hajtott végre.

5.2. Fogalmak

FLOP

Texas Hold’Em játékstílusú játékban a szerver kiosztja az első három közös lapot.

TURN

Texas Hold’Em játékstílusú játékban a szerver kiosztja a negyedik közös lapot.

RIVER

Texas Hold’Em játékstílusú játékban a szerver kiosztja az ötödik közös lapot.

Dealer

Magyarul osztó. A játék során az osztógomb, amely a grafikus felületen D betűvel van jelezve. Az óramutató járásával megegyező irányban halad körbe az asztalon, így minden parti esetén más felhasználó tölti be az a osztó szerepét.

6. A póker játékról

A póker [3], mint kártyajáték igen népszerű szerencsejáték. Akár élő tv adásokat is végig lehet követni, ahol hatalmas főnyereményeket osztanak ki a dobogós helyezetteknek. Viszonylag sokfajtája terjedt el szerte a világon, kezdve a klasszikus 5 lapos leosztásokkal egészen az OMAHA játékstíluson át a jól ismert Texas Hold’Em játékstílusig. A játékot 52 lapos francia kártyapaklival játszák, amelyben 4 szín és 13 különböző értékű kártyalap található. A játékstílusok igen különbözők tudnak lenni. Megkölönböztetünk ante, illetve blind alaptétet. Ante alaptét esetén az asztalnál ülő összes játékos előre meghatározott tétet rak be. Blind alaptét esetén megkölönböztetünk kis és nagy vakot, amelyet körönként más és más játékosok rakkák be. minden esetben az osztótól közvetlenül balra ülő játékosnak kell beraknia a kis vakot. Az osztótól kettővel balra ülő játékosnak pedig a nagy vakot. Amíg az alaptétek nem kerültek be az asztalra, addig a ház nem kezdi meg a lapok kiosztását.

A 5. ábrán látható az OMAHA [4] játékstílus. A Texas Hold’Em játékstílushoz igen hasonló pókerjáték változat. A játékosok a kezükbe négy darab kártyalapot kapnak a vakok betétele után, majd megkezdődik az első licitkör, amelynek a végén három közös lap kerül ki az asztal közepére. Újabb licitkör kezdődik, amelynek a végén egy, majd az újabb licitkör végén, még egy újabb kártyalap kerül az asztal közepére, mint közöslap. A felhasználóknak maximum öt lapot lehet felhasználniuk a legjobb kéz előállítására. Az öt lapból kötelezően kettő a saját kézből történik a maradék három pedig a közös lapokból. Értelemszerűen a legerősebb kéz nyer [5]. A játék célja, hogy minél több zsetont gyűjtsünk össze a partik során.

5. ábra. OMAHA póker játék



6.1. Játékmenet

Minden játékszerver úgy lett konfigurálva, hogy két játékos esetén a parti elkezdődjön. Ha valaki később csatlakozott az asztalhoz, az a megkezdett partiból semmit nem érzékel, mintha üres asztalnál ülne. Csak a következő partiba tud beszállni. Mindkét játékstílus esetén van egy BLIND kör, amikor a szerver bekéri a vakokat a játékosoktól. Az osztótól eggyel balra ülő játékos köteles betenni a kis vakot, a kis vaktól eggyel balra ülő játékos pedig köteles betenni a nagy vakot. Ebből a felhasználó nem lát semmit, egy automatizált eljárás hajtja végre a vakok beadását.

Az osztógomb a legelső körben kerül kiosztásra a legelsőként csatlakozott játékoshoz. Az osztógomb az óramutató járásával megegyező irányban halad. minden új megkezdett parti esetén az osztógomb a következő játékoshoz kerül. minden parti legelső körében a kezdő játékos az osztótól balra ülő harmadik játékos, minden további kört a kis vakra kötelezett játékos kezd meg.

A játékszerverek nem képesek kezelni, ha egy játékosnak elfogyott, illetve nincs elegendő zsetonja. Ebből kifolyólag esetenként előfordulhat negatív egyenleg, illetve nem definiált viselkedés. A játékosok szigorúan csak egymást követve küldhetnek utasításokat a szervernek. A felhasználói grafikus felületen egyértelmű jelzéssel van ellátva az éppen soron levő játékos. A játékosok megadhatják, emelhetik a tétet, illetve, ha nem szeretnének az adott körben semmit csinálni, akkor CHECK típusú üzenetet is küldhetnek a szervernek. Az emelés mértéke a mindenkorai játékasztal alaptét felét jelenti. Ugyanakkor lap eldobásra és a játék elhagyására is lehetőség van. A felhasználók korlátozottan visszanézhetik a korábbi leosztásokat, és a partiban történt eseményeket. A játékosok asztalonkénti maximum száma 5 fő. A kliensek a játék elhagyását követően újracsatlakozhatnak az adott játékszerverre a fentieket figyelembe véve. Lehetőség van asztalt váltani, és a játékszerverek korlátozottan képesek kezelni, ha a játékossal megszakad a kapcsolat. A kliens alkalmazások is korlátozott mértékben fel vannak készítve az esetleg kommunikációs hibákra.

6.2. Játékstílusok

A póker játék meglehetősen sok játékstílusban terjedt el szerte a világon. Régebben még a kártyapakli is eltért a ma használatos 52 lapos francia kártyapaklitól. A játékstílusok közül kettőt valósít meg a programcsomag

- Classic [6]
- Texas Hold’Em [7]

A két játékstílus közötti legfőbb különbség, hogy a klasszikus játékstílus esetén nincsenek közös lapok, míg Texas Hold’Em játékstílus esetén vannak.

Classic

A klasszikus játékstílus legfőbb ismertető jele, hogy mindenki öt lapot kap kézbe és nincsenek közös lapok. Először is a vakokra kötelezett játékosok automatizált formában rakják be a vakokat, azután pedig úgynevezett előkör van, amikor a szerver minden játékosnak 5-5 darab kártyalapot oszt kézbe és ezek alapján lehet licitálni. Ha vége a körnek, akkor mindenki kicsérélheti a lapjait, amiket saját maga választ ki a grafikus felületen a saját kártyalapjainak rákattintásával. Ha a kártyalap feljebb csúszott a grafikus megjelenítésen, akkor a kártyalapot cserére jelölte a játékos. A

Change feliratú gombbal cserélhetőek a kártyák. mindenki akkor kapja meg az új kártyalapjait, ha már mindenki nyilatkozott a cseréről. Ezek után új kör indul, amikor is az új lapok birtokában tehetik meg a tétjeiket a játékosok. A kör után a szerver kihirdeti a nyertest, és a nyertes lapokat az asztal közepén jeleníti meg a grafikus felület. Kivétel, ha az éppen adott felhasználó a nyertes, akkor a nyertes kártyalapok maga előtt jelennek meg, íyenkor más kártyalapok nem kerülnek felfordításra. A felhasználók a nyertes lapok megtekintését követően kötelesek rágattintani a Check feliratú gombra, illetőleg ha kifutnak az időből, akkor a játék asztal kilépteti őket. Ha ebben a körben is mindenki nyilatkozott, akkor a játékasztal új partit indít.

Texas Hold’Em

A klasszikus játékstíllussal erősen megegyező játékmenetű stílus. A különbség csupán annyi, hogy a ház 2-2 darab kártyalapot oszt minden játékosnak a kezébe, amelyekkel a játékosok a parti végéig rendelkeznek. Illetőleg a nyertes lapok semmilyen esetben sem középen, hanem a játékosoknál jelennek meg.

7. Futtatás

7.1. A póker szerver elindítása

Az X:\poker\release nevű mappában található meg a poker-server-1.0.0.jar fájl. Nyissunk egy terminált a kijelölt könyvtárban, és adjuk ki a

```
java -jar poker-server-1.0.0.jar
```

parancsot. Ha a 6. ábrának megfelelő konzol naplózást látunk, akkor a szervert sikeresen elindítottuk.

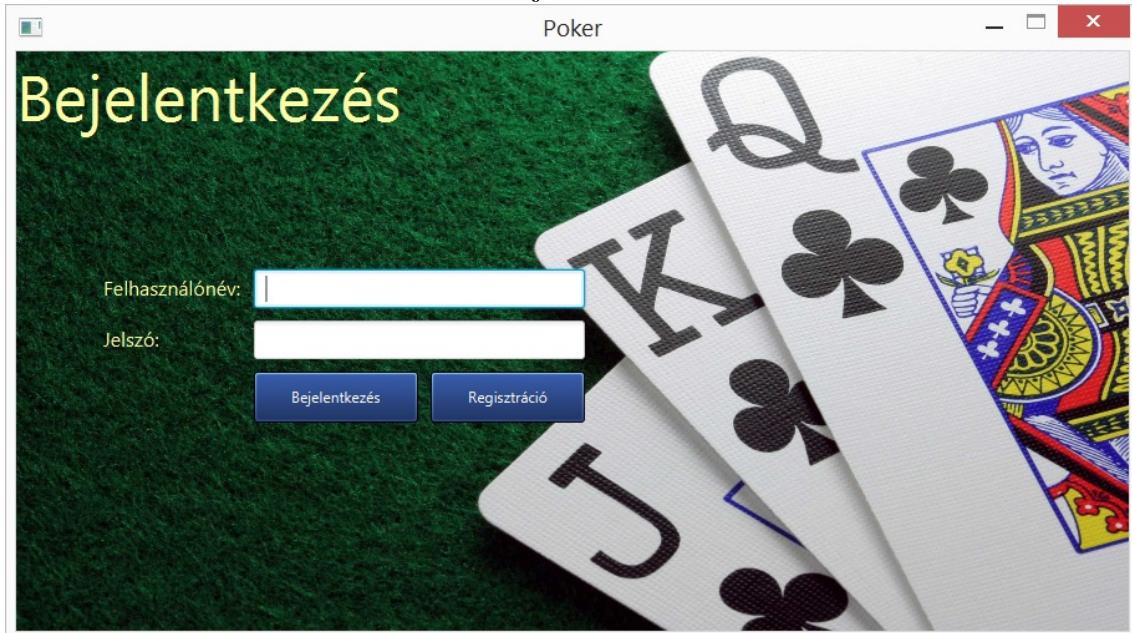
6. ábra. Szerver naplázása

```
D:\projects\szakdolgozat\poker\release>java -jar poker-server-1.0.0.jar
***POKER SZERVER***
Port:1099
Szerver név: pokersv
A szerver elindult
```

7.2. A póker kliens elindítása

A kliens futtatása hasonló módon történik, mint a szerveré. Navigálunk a X:\poker\release mappába, és a konzolon adjuk ki a megfelelő parancsot. Ha a kliens sikeresen elindult, akkor a 7. ábrának megfelelő képernyőképet kapunk.

7. ábra. Bejelentkezési form



8. A póker játék használata

Ha még nem regisztráltuk magunkat a játékba, akkor kattintunk a Regisztráció nevű gombra. Adjuk meg a regisztrálni kívánt felhasználó nevünket és jelszavunkat, majd regisztrálunk. A szerver értesít minket, hogy a művelet sikeres, vagy sikertelen volt. A 8. ábra egy sikeres regisztrációt szemléltet. Ügyeljünk arra, hogy névütközé-

8. ábra. Sikeres regisztráció

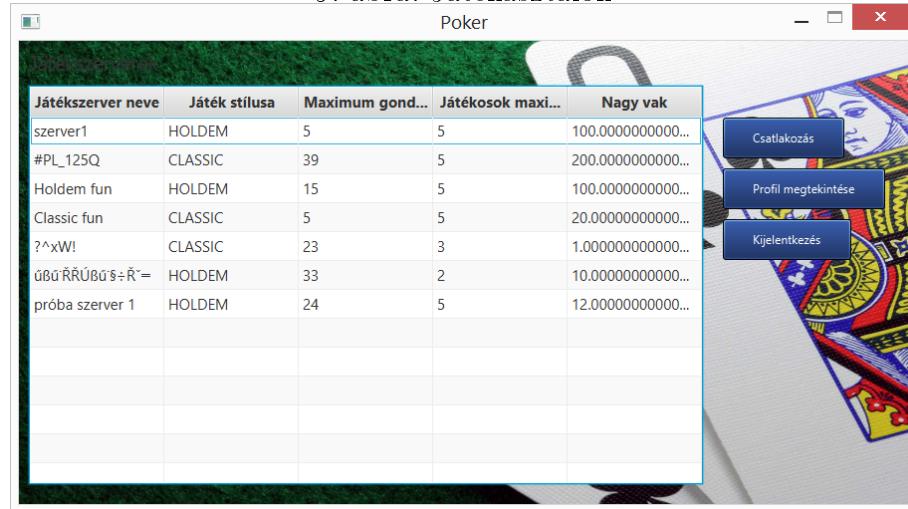


seket a szerver nem enged. Tehát, ha valaki XYZ névvel már regisztrálva van, akkor még egy ugyanolyan névű felhasználót nem enged regisztrálni a szerver. Továbbá

győződjünk meg arról, hogy a megadott jelszavak megegyeznek.

Sikeres regisztrációt követően a program visszairányít minket a bejelentkezési formához, amelyet értelemszerűen kitöltve be tudunk jelentkezni a programba. A sikeres bejelentkezést követően a 9. ábrán látható felület fogad minket, ahol például asztalhoz csatlakozhatunk. Tetszőlegesen válasszunk ki egy Texas Hold’Em játékstílusú asztalt, majd kattintsunk a Csatlakozás feliratú gombra.

9. ábra. Játékasztalok



10. ábra. Üres játékasztal



A program átirányított minket a 10. ábrán jelölt felületre. Az üres játékasztal két dolgot jelenthet, vagy azt, hogy az asztalnál éppen játszanak, vagy pedig azt, hogy az asztalnál mi vagyunk az egyedüli játékosok, jelen esetben egyedüliként ülünk az asztalnál. Indítsunk el egy második kliens programot is, majd a fenti utasításokat követve csatlakozzunk az előbbi játék asztalhoz. Miután csatlakoztunk, azután indítsunk el még további három darab kliens programot és csatlakozzunk az előbbi

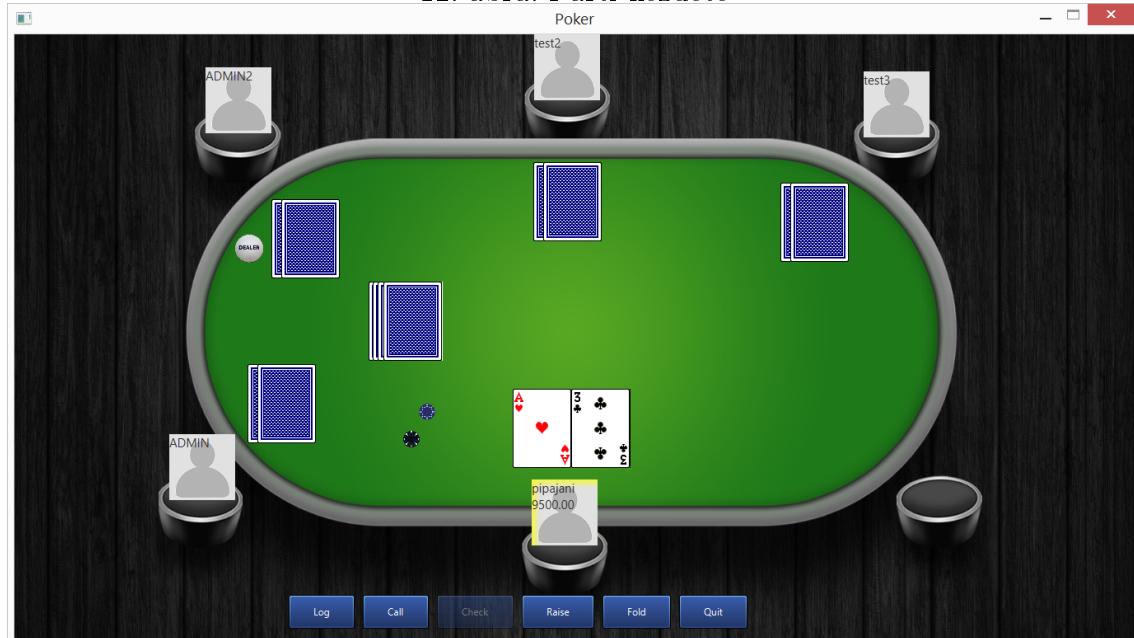
játék asztalhoz. A 11. árbán láthatjuk a szerver naplózását, amelyről leolvasható, hogy az imént csatlakozott három darab felhasználó várólistára került.

11. ábra. Szerver naplózása

```
WAITING: Proxy[PokerRemoteObserver, RemoteObjectInvocationHandler
ote], objID: [-1128ed17:1516c52d1fc:-7ff9, -1318139461981679411]]
WAITING: Proxy[PokerRemoteObserver, RemoteObjectInvocationHandler
ote], objID: [-70ec7a1c:1516c52d4d4:-7ffa, 5285224842646616625]]
WAITING: Proxy[PokerRemoteObserver, RemoteObjectInvocationHandler
ote], objID: [40a3364b:1516c52d868:-7ffb, 7012967986114554691]]]
```

Térjünk vissza a játékban lévő kliens programokhoz és játszuk le a megkezdett partit. A következő parti elindulásakor a várólistán szereplő kliensek teljesértékű játékosként csatlakoznak be az éppen elindult partiba, így az asztalnál a 12. ábra alapján már öt darab játékos fog ülni. Az asztal Texas Hold’Em játékstílusú, így

12. ábra. Parti kezdete



mindenki kettő darab kártyalapot kap kézbe. A kis és nagy vakokat az asztal bekérte, a beadott zsetonok meg is jelentek az asztalon. Jelenleg pipajani következik. Az ADMIN2 nevű játékosnál van az osztógomb, tehát a test2 nevű játékos a kis és a test3 nevű játékos pedig a nagy vak. Pipajaninak az asztal felé nagy vaknyi tartozása van, amelyet az egyenlege alapján meg tud adni.

A 13. ábrán látható, hogy a szerver már kiosztotta mind az öt darab közös lapot és a még senki sem dobta el a lapjait. A 14. ábrán már az látható, amikor a parti hamarosan véget ér. A szerver ebben a körben hirdette ki a nyertes lapokat a játékosok között. A test3 nevű játékos eldobta a lapjait, azonban értelemszerűen ő is láthatja a nyertes lapokat. Jelenleg a nyertes test2 nevű játékos következik. Ha az asztalnál ülő partiban maradt játékosok mindegyikre megnyomta a Check feliratú

13. ábra. RIVER



gombot, akkor a játékasztalon új parti kezdődik. A várakozólistán szereplő játékosok teljes értékű játékosként szerepelnek az asztalnál, továbbá azok a játékosok is kapnak kártyalapokat, akik az előző partiban eldobták a lapjaikat.

14. ábra. Nyertes lapok kihirdetése



III. rész

Fejlesztői dokumentáció

9. Felhasznált technológiák

A fejlesztés során fejlesztőkörnyezetet használtam, így viszonylag sok probléma már a fordítás és futtatás előtt kiderült. Ahogy korábban is leírtam, a programcso-magnak szüksége van MySQL Serverre az adatbázis kérdése miatt. A kódot verziókkal is el kellett látnom, így egy verziókezelő rendszert is felhasználtam. Az osztály- és moduldiagramok szerkesztésére ugyancsak külső szoftvercsomagot használtam fel. A programcsomag több modulból épül fel, ezért ugyancsak szükségessé vált egy build-rendszer használata, amely a modulok közötti függőségek kezeléséért felelős.

9.1. Eclipse

A szakdolgozatom eclipse Mars [8] fejlesztőkörnyezetben írtam, amelyhez a quick search [9] nevű plugint is letöltöttem. A fejlesztést nagy mértékben megkönnyítette a fejlesztőkörnyezet használata, ugyanis többek között szintaktikus és classpath ellenőrzést is végzett. Továbbá remekül paraméterezhető jóformán tetszőleges bilentyűkombinációkkal.

9.2. Maven

A népszerű buildrendszerből a 3.3.3-as verziót használtam, amelyhez igen sok különböző plugin szerezhető be. Eredetileg csupán a függőségek kezelésére alkalmaztam volna a programot, később egyre több és több pluginnal egészítettem ki, így egyre több feladatot volt képes ellátni a szoftver.

Adatbázis séma létrehozása és demó adatokkal való feltöltése

Az adatbázis sémát a fejlesztés elején minden manuálisan készítettem el, majd rátaláltam a sql-maven-plugin nevezetű pluginra. A maven honlapján megtalálható teljeskörű dokumentáció segítségével bekonfiguráltam a pluginot, így csupán a

```
mvn sql:execute@create-db  
mvn sql:execute@create-triggers  
mvn sql:execute@fill-db
```

utasításokat kellett kiadnom parancssorból, hogy friss adatbázissal dolgozhassam tovább. A felesleges parancsmétlések elkerülése végett letöltöttem az exec-maven-plugin nevű pluginot, amelynek segítségével sikerült a

```
mvn exec:exec
```

parancsra korlátozni az adatbázis újboli létrehozását. A feladatot tovább általánosítottam, majd végül egy profilt készítettem el, amely az install fázissal együtt oldja meg az adatbázis kérdését. A

```
mvn clean install -Pbootstrap
```

parancsal nem csak a lokális repozitorinkba telepíthetjük fel a poker-persist modult, de még egy új adatbázist is kapunk.

Javadoc

A mavennel java dokumentációt is lehet generáltatni. Navigálunk a poker könyvtárba és adjuk ki a

```
mvn javadoc:javadoc
```

parancsot. Aggregált java dokumentációra is lehetőség van a

```
mvn javadoc:aggregate
```

Sőt, akár modul dokumentációt is lehet készíteni a

```
mvn site:site
```

```
mvn site:stage
```

utasításokkal. Az elkészült dokumentációk a modul `poker\target\staging` könyvtárba kerülnek. Kettő darab index.html állományt kell keresnünk, az egyik a gyökerkönyvtárban található a másik pedig az `apidocs` könyvtár gyökerében.

9.3. MySQL

Az adatok perzisztens tárolásáért egyértelműen relációs adatbázisra volt szükség. A MySQL adatbázissal már korábban is találkoztam, ismert technológiát szerettem volna használni.

9.4. Git

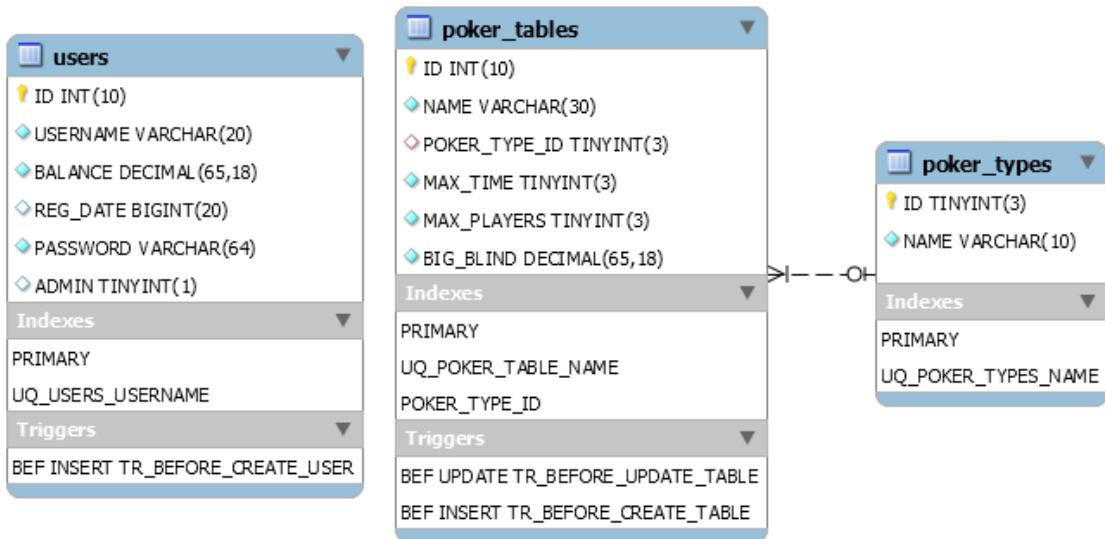
A fejlesztés során előfordult, hogy internet kapcsolat nélkül kellett verzióznom a kódomat. A git egyik legnagyobb előnye, hogy internet kapcsolat nélkül is lehet commit parancsot végrehajtani. A commit parancs hatására a lokális repozitorinkba egy új pillanatfelvétel készül az aktuális munkakönyvtárunk tartalmáról, amelyet később push utasítással feltölthetünk egy távoli repozitoriba. A fejlesztéshez több branchet is használtam, így ha valami balul sült volna el, akkor egyszerűen visszaálltam volna a masterre.

9.5. Enterprise Architect

A tervezés során igen nagy hasznát vettettem ennek a szoftvernek. Rengeteg beállítás, funkció megtalálható a programban. Szinte minden testreszabható az UML diagramokat érintő kérdések körében. Az elkészült UML diagramokból pedig JAVA kódot tud generálni a program. A dokumentációban fellelhető osztály hierarchiák és UML diagramok ennek a szoftvernek a segítségével készültek el.

10. Adatbázis séma

15. ábra. Adatbázis séma



Az adatbázis három darab táblából épül fel, amelyet a 15. ábra szemléltet.

10.1. Póker játék asztal tábla

Oszlopnév	Típus	Megszorítás
ID	UNSIGNED INT	Elsődleges kulcs a táblára nézve.
NAME	VARCHAR	Az oszloban szereplő értékeknek egyedinek kell lenniük és legfeljebb 30 karakter hosszúak lehetnek.
POKER_TYPE_ID	UNSIGNED TINYINT	Az oszlop idegen kulcs a poker_types táblára nézve.
MAX_TIME	UNSIGNED TINYINT	Az értéknek bele kell esnie a [5-40] intervallumba.
MAX_PLAYERS	UNSIGNED TINYINT	Az értéknek bele kell esnie a [2-5] intervallumba.
BIG_BLIND	DECIMAL	-

10.2. Felhasználók tábla

Oszlopnév	Típus	Megszorítás
ID	UNSIGNED INT	Elsődleges kulcs a táblára nézve.
USERNAME	VARCHAR	Az oszloban szereplő értékeknek egyedinek kell lenniük és legfeljebb 20 karakter hosszúak lehetnek.
BALANCE	DECIMAL	-
REG_DATE	BIGINT	-
PASSWORD	VARCHAR	Az értéknek pontosan 64 hosszúnak kell lennie.
ADMIN	UNSIGNED TINYINT	Az érték csak 0 vagy 1 lehet.

10.3. Játékstílus tábla

Oszlopnév	Típus	Megszorítás
ID	UNSIGNED INT	Elsődleges kulcs a táblára nézve.
NAME	VARCHAR	Az oszlopan szereplő értékeknek egyedinek kell lenniük és legfeljebb 10 karakter hosszúak lehetnek.

Minden tábla rendelkezik elsődleges kulccsal, amelynek típusait a fentebb látható táblázat tartalmazza. A `poker_types` tábla elsődleges kulcsa UNSIGNED TINYINT, melyet a MySQL Server 4 byteon tárol. Az adatábrázolás mértékének a szűkítése ebben az esetben indokolt, ugyanis a 255 különböző értékű UNSIGNED TINYINT típus kielégíti a játéktípusok által támasztott követelményeket.

Az adatbázis sémában három darab trigger van definiálva:

- Tábla beszúrás előtti
- Tábla módosítás előtti
- Felhasználó beszúrás előtti

11. Megoldási terv

11.1. Probléma

Egy hálózaton keresztül működő póker játékot szolgáltató kliens-szerver programcsomagot fogok megvalósítani JAVA programozási nyelven. A felhasználói és szolgáltatással kapcsolatos adatokat egy adatbázisban helyezem el, így is biztosítva a játék menet konzisztenciáját és konkurencia vezérlését. Az említett adatabázis többek között rögzíti az olyan profil adatokat, mint a felhasználói név, a jelszó vagy a regisztráció dátuma. Az alkalmazás szerver-oldali komponensei több olyan asztal szimultán futtatását biztosítja, amelyek különböző paramétereiket lehetnek. Ugyanis felhasználói oldalról különböző igények léphetnek fel (pl.: a tapasztaltabb játékosok rövidebb idő alatt tudnak reagálni egy leosztásra). Az asztalok leglényegesebb paraméterei a játék stílus, a játék gyorsasága, illetve az elfoglalható helyek száma lesz, de ezeken felül további paraméterek is megadhatóak egy konkrét asztal instancia létrehozásakor. Ezen műveletek ellátására, azaz az asztalok konfigurálására, egy grafikus interfész valósít meg a programcsomag.

A játékban a felhasználók jogok alapján is megkülönböztethetőek lesznek, így a magasabb jogkörrel rendelkező felhasználók speciális műveleteket hajthatnak végre a játékon és az adatbázison egyaránt. Ebből kifolyólag külön adminisztrációs felület is megvalósításra kerül majd, amelyen keresztül adott esetben akár rendezni (név, regisztráció dátuma, stb. alapján), ill. törölni is lehet majd felhasználókat, sőt a különböző felhasználóknak különböző jogokat is itt lehet majd kiosztani. Továbbá a felhasználók könnyen tudják majd menedzselni a saját felhasználói fiókjukat, ugyancsak egy grafikus interfész segítségével.

11.2. Tervezés

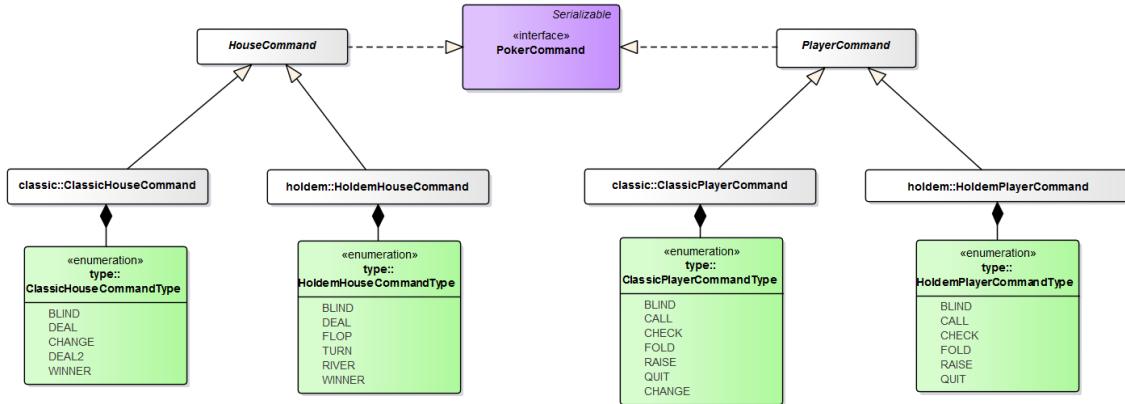
A probléma leírásából fakadóan egyértelműen kiderül, hogy legalább kettő darab modulra lesz szükségünk, egy kliens modulra és egy szerver modulra. A tervezés során végig kellett gondolni, hogy mik azok az igen fontos események amelyek a program futása során bekövetkeznek, illetve bekövetkezhetnek. A szerver és a kliens hálózaton keresztül fognak kommunikálni, így kínálkozik a lehetőség az RMI architektúrára való építkezésre. Ha RMI, akkor egy jóldefiniált publikus interfész kell elkészíteni, amely a szerver vázáért felel. A kliensek ezen az interfészen keresztül hívnak be a szerverhez. A probléma megfogalmazásából kiderül, hogy melyek azok az alapvető funkciók és szolgáltatások, amelyeket a szervernek biztosítania kell. Ezen megfogalmazás alapján már körvonalazódik a szerver modul feladata. A probléma leírásából a szerver és a kliens között kiépített kommunikációs rendszer csak implicit megfogalmazásban szerepel. Az, hogy hogyan és miként, illetőleg milyen szekvencia szerint fognak egymással kommunikálni a modulok az explicite nincs felvázolva. Az implementáció során ügyelni kell arra is, hogy párhuzamosan akár több kliens is csatlakozhatva lehet a szerverhez, illetve az egyes játék asztalokhoz. Az üzenetek egy előre meghatározott sorrend alapján küldhetőek és fogadhatóak. Fontos megjegyezni, hogy nem csak a kliensek küldhetnek üzeneteket a szervernek, hanem a szerver is értesítheti a klienseket a szerveren történő eseményekről, mint például, ha egy adott asztalnál egy játékos CALL utasítást küld a szervernek, akkor a szerver az asztalnál ülő játékosokat értesíti a bekövetkezett eseményről. Ennek megfelelően a kliensek minden értesítést kapnak az őket érintő eseményekről. Ezáltal a kliens modulnak is meg kell fogalmazni egy publikus interfész, amelyet a szerver fog felhasználni. Fontos megjegyezni, hogy a kliensek közvetlenül egymásnak nem küldhetnek üzeneteket. A szervert úgy kell implementálni, hogy a bérkező üzenetek alapján az összes megfelelő klienst értesítse.

Az üzeneteknek kettő fajtáját különböztetjük meg

- Szerver által küldött üzenet
- Kliens által küldött üzenet

A megfelelő absztrakció elérése érdekében a 16. ábrán látható `PokerCommand` interfésszel össze kell fogni az üzenet fajtákat. A 16. ábráról leolvasható, hogy

16. ábra. Póker üzenetek felépítése



minden üzenetnek van egy típusa. Az üzenet típusa függ az üzenet fajtjától és specializációjától is. Például ha egy Texas Hold’Em játékstílusú asztal be szeretné kérni a vakokat a játékosuktól, akkor egy `HoldemHouseCommand` típusú `HoldemHouseCommandType`::`BLIND` utasítástípusú új objektum példányt kell szétküldenie az asztalnál ülő összes játékosnak. Ezt az üzenetküldést a szerver a kliensek publikus interfészén keresztül teheti meg.

A kliensnek az üzenetet fogadnia kell. Az üzenet fogadására egy új, ezt a funkciót ellátó objektumot kell definiálni.

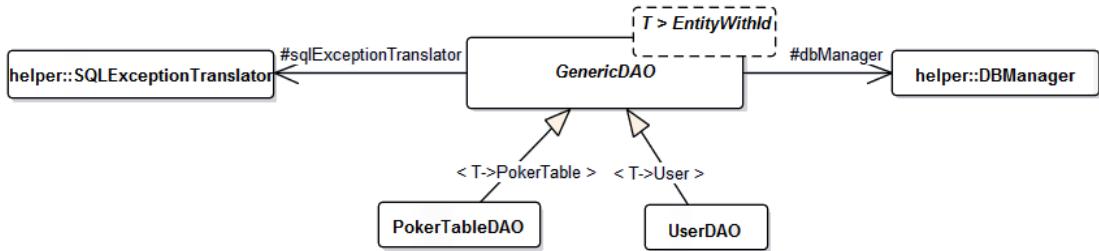
A kliens oldal architektúrális felépítésének az alapját a Model-View-Controller tervezési minta adja. A beérkezett üzenetet az erre kijelölt objektum fogadja, majd továbbítja a kliens oldali vezérlő rétegnek. A vezérlő réteg értelmezi a bejövő üzenetet, majd ennek megfelelően a model rétegnek és a megjelenítési rétegnek is továbbítja a beérkezett üzenetet további feldolgozásra. A probléma különböző funkciók leírását is tartalmazza. Célszerű, hogy a különböző funkcióknak különböző grafikus interfésszt biztosítsunk. Ugyanakkor új felület létrehozásakor új vezérlő objektumot is létre kell hoznunk, viszont a model rétegnek statikusnak kell lennie. Ugyanaz a model objektum lássa el például a bejelentkezési és regisztrációs kérelmeknek a továbbítását a szerver felé. Tehát különböző funkciók elérése esetén a vezérlési és megjelenítési réteget ki kell cserélni, viszont a modelt nem. Illetőleg kettő darab modelt kell megfogalmaznunk, az egyik látja el például a fentebb említett egyszerűbb funkciók meghívását a szerveren, a másik pedig a tényleges játék közbeni model, amelyet a vezérlői réteg értesíteni tud a beérkezett üzenetekkel kapcsolatban és kliens oldalon biztosítja a játék menetét.

A probléma leírásában szerepel adatbázissal való kommunikáció is. Erre ugyancsak egy új modult kell bevezetni, amely az adatbázissal történő kommunikációt bonyolítja le. Entitás osztályokat írni és olvasni kell az adtbázisból, így ezeket az osztályokat

általánosan kell megfogalmazni a kód duplikációk elkerülése végett. Ezen modul vázlatát a 17. ábra szemlélteti.

Az adatbázissal való kommunikációért felelős modul nem csak entitáso-

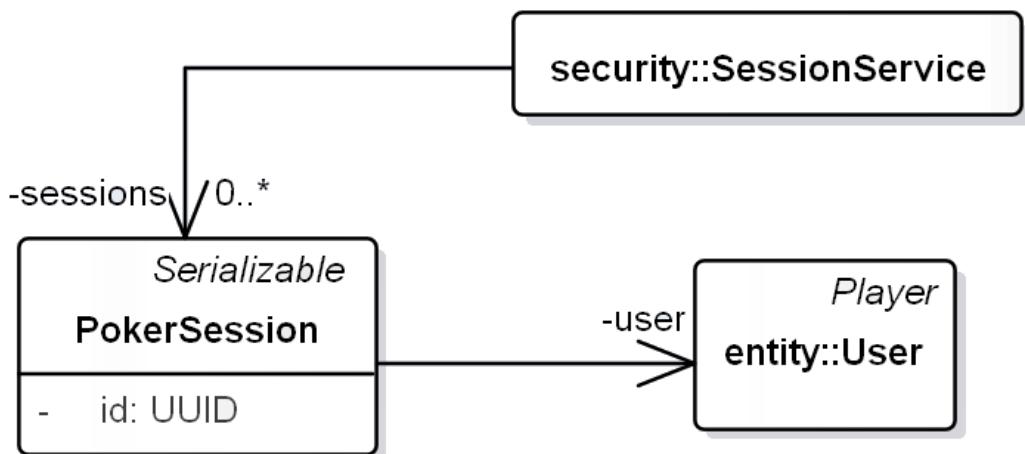
17. ábra. Az adatbázissal kommunikáló modul



kat tartalmaz, hanem további kettő darab objektumra is szükség lesz. Az SQLExceptionTranslator fontos része a modulnak. Az adatbázis triggerek és megszorítások által definiált hibaüzeneteket fordítja át JAVA kivételek objektumokra, amelyek a kódbeli feldolgozást könnyítik meg, továbbá ezek a saját kivételek objektumok már a felhasználó számára is értelmes hibaüzenetekkel bírnak. A DBManager osztály felelős az adatbázissal kiépített kapcsolat fenntartásáért és lezárásáért.

A szervernek szerves része egy objektum, amely a munkamenetek kezeléséért felelős. Az autentikált felhasználók egy-egy munkamenettel azonosíthatóak. Az autentikációért felelős részmodult a 18. ábra vázolja fel. A SessionService tartja nyilván

18. ábra. A munkamenetek tárolása szerver szinten



a munkameneteket. Egy felhasználóhoz legfeljebb egy munkamenet rendelhető egy adott időben. Továbbá egy munkamenethez pontosan egy felhasználó tartozik. Ha a

felhasználó kijelentkezik a póker játékból, akkor a felhasználó munkamenetét érvényteleníteni kell. Ugyanaz a munkamenet többször nem osztható ki, törölni kell. Ugyanakkor egy felhasználó különböző időpillanatokban rendelkezhet különböző munkamenetekkel. Tehát ha az imént kijelentkezett felhasználó újból be szeretne jelentkezni a játékba, akkor egy új munkamenettel kell azonosítani. A szervernek biztosítania kell, hogy a kliensek rendelkezzenek munkamenet objektumokkal, ugyanis bármilyen funkció hívása esetén a kliensek ezzel az objektummal azonosítják magukat. Pontosabban fogalmazva minden munkamenet egy-egy UUID értékkal van ellátva, és a klienseknek elég csak ezt az egyedi értéket elküldeni a szervernek. A felhasználóknak nem engedhetjük meg, hogy egy felhasználónévet konkurensten használhassanak. Ez a felügyelet ugyancsak a `SessionService` objektum feladata.

A szerver és a kliens közötti bárminemű tevékenység, illetve kommunikáció egy közös modult kell, hogy felhasználjon. Például ha kivételobjektumot szeretnénk dobni szerver oldalon és azt kliens oldalon szeretnénk kezelni, akkor értelemszerűen minden szerver minden a kliens classpathján szerepelnie kell az adott kivételobjektumnak. A közös modulban kell szerepelnie a szerver és a kliens publikus interfészüknek, a kommunikációs rendszert alkotó osztályok összessége és a saját kivételek osztályok definíciói.

Külön modul fogalmazható meg a játékosok lapjainak a kiértékelésére is. Ez a modul nem kerül implementálásra, külső könyvtárként kerül felhasználásra.

11.3. Implementálás

Az implementálást a kliens oldali megjelenítéssel kezdtem egy maven archetype generator segítségével. Így generálódott egy megfelelő kiindulási alap, amiből elkezdtettem építkezni. Legelőször a bejelentkezési és a regisztrációs formot állítottam össze. Ezekután pedig a szerver publikus interfészének egy részét fogalmaztam meg kód szinten. Ezt az interfészt a kliens már el tudta kérni az RMI registry-ból [15], és sikeresen be tudott hívni a szerverhez. A szerver és a kliens kommunikációjának az alapjának a kiépítése után az adatbázis feltelepítése, elindítása majd elérése volt a legfőbb céлом.

Miután sikerült promptból becsatlakoznom a MySQL Serverre, azután manuálisan létrehoztam egy kezdetleges adatbázis sémát és néhány egyszerű INSERT műveletet hajtottam végre az adatbázison. A kezdetleges demó adatokat a szerver oldali kóddal `PreparedStatement` objektumok segítségével olvastam ki. Ugyanezen objektumok segítségével írtam az adatbázisba. A későbbiek során kialakult a jóformán végleges adatbázis séma, amelyet egy sql állományban tároltam el, és ezt az állományt pusztán alkalmazni kellett az adatbázisra, ha új adatbázis sémát szerettem volna elérni. A későbbiek során az adatbázisba letárolt értékekre megszorításokat kellett

alkalmaznom, amelyet akár triggerekkel, akár constraintekkel valósítok meg. Így az adatbázis sémát generáló sql állomány mellé létre kellett hoznom egy olyan sql állományt, amely triggereket aggat a meglévő adatbázisbeli adattáblákra. Továbbá a fejlesztés haladtával demó adatokat is kellett biztosítanom az adatbázis számára, ezzel is lerövidítve a fejlesztési időt.

Miután a szerver már képes volt az adatbázissal való kommunikációra, azután a már meglévő sql kódok és a tervezési fázisban felvázoltak alapján a poker-model modult el lehetett készíteni. Szükség volt egy közös interfészre, amely összefogja az entitásokat. Az interfész alapján el lehetett készíteni a `GenericDAO` osztályt, amely típusparaméterrel van ellátva. A típusparaméterre vonatkozó kritérium, hogy az `EntityWithId` interfész implementálnia kell a paraméterül adott típusnak. A poker-persist modulnak rendelkeznie kell egy kapcsolatkezelő objektummal is. Ennek megvalósítása nem jelentett különösebb gondot, viszont a kapcsolat paramétereire utólag került sor. A már megírt triggerek alapján az `SQLExceptionTranslator` osztály implementációja ugyancsak könnyűnek mondható. A tervezett 6 darab modulból 2 darab már implementálásra került.

Mivel a szerver és a kliens közötti kapcsolat kiépítésre került, továbbá a szerver már az adatbázist is eléri, ezért a felhasználók bejelentkezése és regisztrációja következett. Ezen funkciók lefejlesztése után a további funkciók kerültek implementálásra. Többek között a játékasztal entitások létrehozása, módosítása, már meglévő felhasználók jelszavának módosítása, felhasználók törlése, adminisztrációs jogok kezelése. A fejlesztési folyamat itt erősen megakadt, ugyanis nem tartottam egyszerűnek a tényleges póker játék szerverek implementációját, és a hozzájuk tartozó kliens oldali kód megírását. Így a már meglévő funkciókat kezdtem el tesztelni és tovább fejleszteni. Lefejlesztésre került a `SessionService` osztály is, amely a kliensek munkameneleinek kezeléséért felel. A fejlesztés korai fázisában volt egy hetedik, poker-security modul, amely a kliensek autentikálását és autorizációját végezte volna. Azonban a fejlesztés előrehaladtával és a kód refaktorálásával a modult fel kellett számolni és a modul végül egyetlen osztályként szerepel a poker-server modulban.

Az eddigi lefejlesztett funkciókon további fejlesztési lehetőség nem kínálkozott a tervezési fázisban megfogalmazottak alapján. A póker játék asztal szerverek implementációja következett. Legelőször a kliens oldali grafikus interfész fejlesztettem le. Viszonylag hamar rá kellett jönnöm, hogy szükségem lesz egy olyan osztályra, amely a játék és a grafikus felület konstans értékeit szolgáltatja. Az fxml állományok nem kerülnek kifordításra, így ha módosítani kellett őket, akkor nem kellett a futó programot újraindítani, elég volt az adott fxml állományt újra betölteni, és a változások érvénybe léptek, így a fejlesztés ugyancsak felgyorsult. A póker asztalok pontos működésének a megfogalmazása és implementálása volt az első olyan pont a fejlesztés során, amelyet többször át kellett dolgoznom. A tervezési fázisban felvál-

toztak kód szinten közel sem triviálisnak mondható dolgok. A póker játék szerver implementálása során absztrakciós szintre nem gondoltam, így először minden a Texas Hold’Em játékstílus követelményei alapján került implementálásra. Ezután egy refaktorálás oldotta meg a fejlesztési fázis továbblépését, mind szerver, mind kliens oldalon. A refaktorálással és a hiányos tervezéssel viszonylag sok idő elúszott, pláne a Test Driven Development megközelítés mellőzése miatt is. Az eddig implementált funkciókat és működésbeli viselkedéseket refaktorálás után és közben folyamatosan manuálisan kellett tesztelni.

A refaktorálást követően nekiállhattam a klasszikus játékstílus implementálásának. Az implementálás során további osztályok absztrakciójára volt szükségem, így még több refaktorálást kellett elvégeznem a kódbázison. A refaktorálást követően a kódbázis sokkal letisztultabbá vált. Ugyanakkor az utasítások típusait enum objektumokként definiáltam, így az utasításokat nem tudtam kellően általánosan megfogalmazni. Később az utasítások típusait egy interfésszel összefogtam, majd a **GenericDAO** mintájára típusparaméterrel láttam el a **PokerCommand** interfész. A megközelítés sikertelenül zárult. A típusparamétert eggyel lejjebb szintre süllyesztettem, tehát a következőkben a **HouseCommand** és **PlayerCommand** osztálydefiníciókat láttam el típusparaméterrel, és az utasítások típusait másmilyen formában csoporthoztattam. A második megközelítés ugyancsak sikertelenül zárult. Ezekután az enumokat felszámoltam és külön osztályokat definiáltam az utasítások típusainak. Már az első lépéseknel kiderült, hogy a kódbázis jelentős részét át kéne alakítani az előzőleg felvázolt elképzelés integrációjához. Ismét sikertelen kísérlet. Viszonylag sok időt emészttettek fel a próbálkozások, ugyanakkor szerencsére a verziókezelő rendszer nagyhasznomra vált. Könnyen vissza tudtam állni korábbi commitokra, illetve a branchek használata is előtérbe került.

Az implementáció az eredeti, enum alapú megoldásra támaszkodik, ezáltal kód duplikáció figyelhető. A kód duplikációk egészen a póker játék asztal szerverekig gyűrűznek fel. Az **AbstractPokerTableServer** jónéhány absztrakt metódusának implementációja csupán 1-1 sorban tér el, ez pedig a kérdéses enum objektumok értékkedása.

11.4. Elemzés

A tervezési fázisban viszonylag sok dolgot sikerült átgondolnom, ugyanakkor kód duplikációk figyelhetők meg a kódbázisban szerver és kliens oldalon egyaránt. A fentebb vázoltak alapján teljesmértékű absztrakció nem valósult meg.

12. Modulok

A programcsomag 6 darab főmodult tartalmaz

poker-server

A póker játék szervere, amely magát a játékot szolgáltatja. A kliensek ennek a modulnak egy instanciájához csatlakoznak.

poker-client

A póker játék kliense, amely segítségével a szerverhez lehet csatlakozni.

poker-shared

A póker játék azon modulja, amelytől a szerver és a kliens egyaránt függ. Olyan közös objektumdefiníciók találhatók itt, mint például az üzenetek és a kivételek leírása.

poker-persist

Az adatok letárolásáért felelős modul. Az adatbázissal lebonyolított kommunikációt teljes egészében elvégzi.

poker-model

A póker játék modellezéséért felelős csomag. Ebben a modulban találhatóak a POJO [11] osztálydefiníciók, amelyeken a poker-persist modul ORM-ét [12] hajt végre.

javapokertexasholdem

Külső könyvtár, amely a nyertes játékos kiértékelési feladatot végzi. Eredetileg csak Texas Hold’Em játékstílus esetén volt képes kezeket kiértékelni, így a modulimplementációt bizonyos mértékben általánosítani kellett, hogy a poker-server modul teljeskörűen fel tudja használni.

A modulok közötti függőséget a 19. és a 20. ábra szemlélteti.

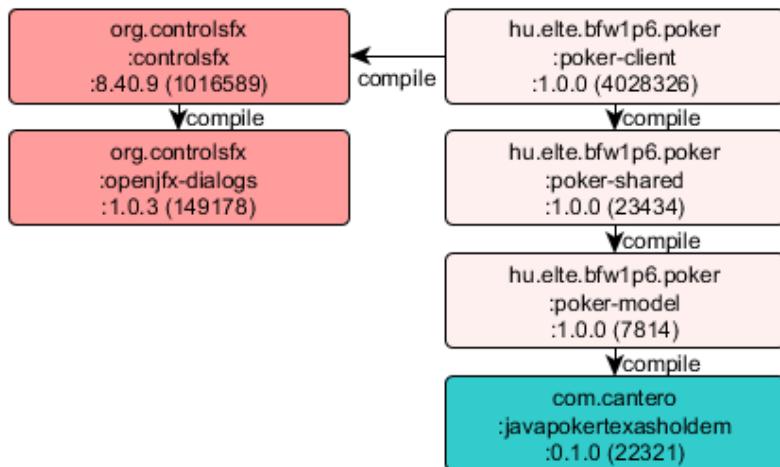
A programcsomag két főmodulra bontható

- poker-server
- poker-client

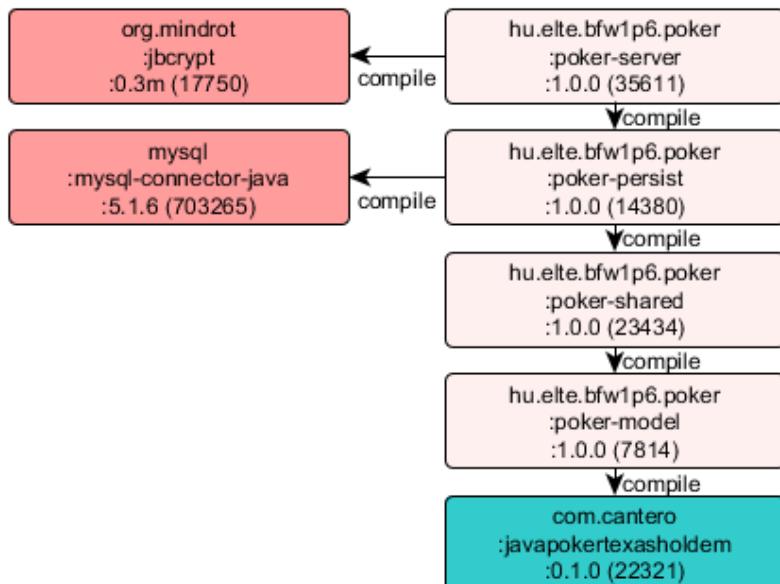
12.1. Model modul

A poker-model modul felelős a játék modellezéséért. A 21 ábrán látható a modul felépítése. Az entitásokat össze kellett fogni az `EntityWithId` interfész segítségével, ezáltal a DAO objektumot generikusan lehetett megfogalmazni, amely segített a

19. ábra. Kliens modulra bontása



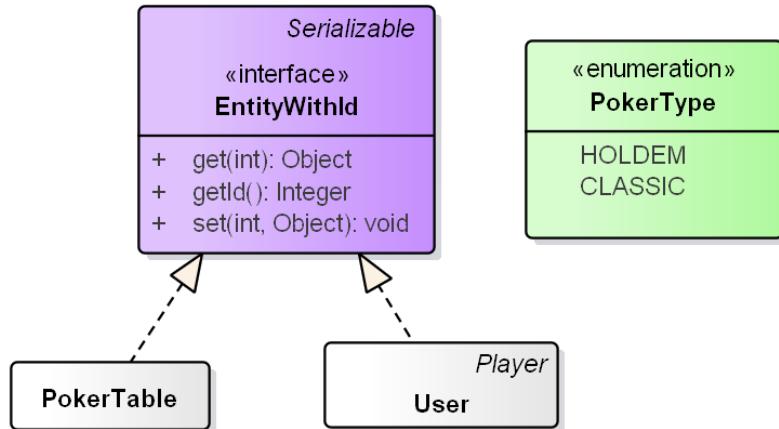
20. ábra. Szerver modulra bontása



kód duplikációk elkerülésében. A `PokerTable` és a `User` osztályok példányai feleltethetőek meg a `poker_tables` és a `users` adattáblák egyes rekordjainak. A `PokerType` enumeráció a játékstílusokat határozza meg. Ha egy új játékstílust szeretnénk hozzáadni a játékhoz, akkor a kódban egy új `PokerType` enum objektumot kell felvennünk, és a `poker_schema.sql` állományban pedig fel kell venni egy új insert utasítást a `poker_types` táblára nézve az új játékstílus nevére vonatkozólag. Természetesen ez a pár sor újonnan hozzáadott kód nem azt jelenti, hogy már használhatjuk is az új játékstílust. A programcsomagot több ponton is ki kell bővíteni

- Szerver oldal

21. ábra. A model felépítése



1. AbstractPokerTableServer osztályt ki kell terjeszteni egy új osztállyal

- Kliens oldal
 1. Új leszármazott osztályokat kell létrehozni a kliens oldali MVC absztrakciós osztályaiból
 2. Új.fxml állományt kell létrehozni

12.2. Shared modul

A poker-shared modul olyan közös osztályokat tartalmaz, amelyeket a kliens és a szerver egyaránt használ. Itt helyezkednek el az egyéni kivételek, a kommunikációs rendszer definíciója és a kliens, mint megfigyelő interfésze.

Kivételek

A játékcsomagnak szüksége van egyéni kivételekre. Az olyan speciális eseteket, mint például adabázisba írás, illetve abból olvasás közbeni fellépő kivételt saját kivételobjektumokkal célszerű lekezelni. Ehhez hasonló egyéni kivételek felléphetnek például bejelentkezéskor, asztalhoz való csatlakozáskor, hibás jelszó megadás esetén.

Kommunikációs rendszer

A szerver és a kliens üzeneteket küldhetnek egymásnak. A kommunikációs rendszer viszonylag absztrakt módon került implementálásra ezzel is elősegítve a későbbi bővíthetőségi nehézségeket. Azonban sajnos helyenként fellelhető kód duplikáció az enumerációkra való építkezés miatt.

Minden utasításnak kötelezően implementálnia kell a `PokerCommand` interfész, ezzel

is elősegítve az általános megfogalmazást a rendszerben. Két fajta utasítás létezik a játékcsomagban:

- Szerver utasítás
- Kliens utasítás

Továbbá a játékstílusnak megfelelően tovább specializálódik a küldendő üzenet fajtája. A programcsomag kettő játékstílust fogalmaz meg 6.2. A játékstílusok különböző utasítás fajták igényelnek. Értelemszerűen, ha egy Holdem játékstílusban résztvevő kliens üzenetet szeretne küldeni a játék asztal szervernek, akkor egy `HoldemPlayerCommand` típusú objektumot kell elküldenie a szerver csonkon keresztül.

Observer pattern [16]

Az oberserver tervezési minta segítségével megvalósítható RMI API-n keresztül az úgynevezett event driven server (magyarul kéne...?). Így nem csak a kliens tud üzenet küldeni a szervernek, hanem a szerver is tudja értesíteni a klienseket. Például, ha egy játékos játékstílustól függetlenül CHECK típusú utasítást küldött a szervernek, akkor azt az üzentet a szerver minden kliensnek szétszórja.

Session

A szerver a klienseket session objektumokkal azonosítja. minden kliens kap egy sessiont a bejelentkezéskor. A session addig él, amíg a felhasználó ki nem jelentkezik, akkor ugyanis a munkamenet érvénytelenítésre kerül. Illetőleg a munkamenetet akkor is érvényteleníteni kell, ha a kliens nem jelentkezett ki, de a kapcsolat valamilyen oknál fogva megszakadt.

Amikor egy kliens be szeretne jelentkezni a játékba, akkor a szerver az összes megszakadt kapcsolatú klienst felderíti, és a munkamenetüket érvénytelennek tekinti. A felderítés úgynevezett pingeléssel történik. A szerver minden csatlakozott klienst megpróbál elérni, és amelyik kliensnél megszakadt a kapcsolat, azt eltávolítja a szerverről. Ezután a `SessionService` ellenőrzi, hogy az adott felhasználónévvel van-e aktív munkamenet, ha van, akkor kivételt dob az eljárás, különben az autentikáció sikeres, és a kliens oldali model eltárolja a loginkor kapott sessiont.

12.3. Persist modul

A poker-persist modul látja el az adatbázissal kapcsolatos teendőket. Ez a modul írja és olvassa az adatbázist a bejövő kérések és paraméterek alapján.

Generikus DAO

A generikusság elengedhetetlen a kódismétlés elkerülése végett. Általánosan kell megfogalmazni az entitások viselkedését 12.1 és ezáltal a persist modul letisztult arcukatot kap. A **GenericDAO** osztály tartalmaz minden olyan elemet, amelyre a specifikálódott DAO-knak szükségük lehet.

Adatbázis menedzser

Az **AbstractDAO**-nak szükséges van tényleges adatbázis kapcsolatra, amelyet a **DBManager** osztály szolgáltat.

Kivételek átfordítása

Az **AbstractDAO** rendelkezik egy **SQLExceptionTranslator** objektummal is, amely az adatbázisból érkező hibát fordítja át **PokerDataBaseException** típusú kivételre. A **PokerDataBaseException** kivétel osztály a felhasználó számára is értelmes hibaszöveget hordoz magában.

12.4. Kliens modul

A kliens modult és annak minden függőségét egy jar fileba csomagolva kell szétterjeszteni a felhasználók között, akik majd ténylegesen használni fogják a programot. A 19. ábra alapján a jar file tartalmazni fog minden olyan osztályt és interfészst, amelyre ténylegesen szüksége lesz a kliens programnak. Az összecsomagoláshoz igénybe vehetjük a mvn clean compile assembly:single parancsot kiadva a poker\release könyvtár alá csomagolódik be a futtatható jar állomány.

```
mvn clean compile assembly:single
```

parancsot kiadva a poker\release könyvtár alá csomagolódik be a futtatható jar állomány.

Model-View-Controller [17]

Az igen köz kedvelt tervezési minta alapján valósítottam meg a kliens oldali programot.

Model

A kliens kettő darab model osztályt tartalmaz. Az **Model** nevű singleton osztály felelős a szerver felé irányuló kapcsolat kiépítésében és fenntartásában. Ez az egyszerű szerver hívásokat végzi el. A kliens indulásakor elkéri az RMI registry-ból a szerver csonkot, és bejelentkezéskor eltárolja a szerver által generált munkamenet objektumot. A model segítségével hívhatóak meg a játék funkcióinak a zöme. Ide sorolható a játék minden olyan funkciója, amely nem játék asztalhoz köthető. Például ennek a

modelnek a segítségével lehet bejelentkezni, asztalt és felhasználót módosítani, vagy akár a játékasztalokat elkérni a szervertől.

A kliens tartalmaz egy másik modelt - `AbstractMainGameModel` -, amely a tényleges játékért felel kliens oldalon. A szerver a parti megkezdésekor leküldi a klienseknek, hogy az adott partiban ki hol ül az asztalnál, hány játékos ül az asztalnál, a játékosok neveit és még néhány inicializációs értéket. Ezen értékek alapján a model el tudja dönteni, hogy az adott kliensnek kötelezően be kell rakni vakot. minden parti elején minden kliensnek az adóssága a nagy vakkal egyezik meg. A további körökben az adósság leróható.

View

A megjelenítésért a JavaFX 2 [18] felel. Az alapkonstrukció, hogy a GUI-n megjelenő objektumok zömét fxml állományokba szervezzük és a megjelenítés színtergráffal történik. minden fxml állományhoz pontosan egy controller tartozik, amely az fxml állomány minden újratöltésekor inicializálódik. A controllert az adott fxml állomány gyökerébe kell bekötni.

A programcsomag által definiált játékstílusok igen hasonlóak, ezért egy `AbstractMainView` osztályban megfelelően kiszervezhetőek a közös részek, majd ennek az osztálynak a specializációiban: `ClassicMainView`, `HoldemMainView` kerülnek megvalósításra a tényleges játékstílusbeli különbségek. Például eltérő különbség, amely a GUI-n is megjelenik, hogy a játékosok hány darab kártyalapot kapnak kézbe, illetve a Texas Hold’Em játékstíussal ellentétben a classic játékmódban nincsenek közös kártyalapok.

Controller

A controller általános szerepét ugyancsak egy absztrakt osztály, az `AbstractMainGameController` tölti be. A közös részek nem csak a megjelenítés és a model szintjén figyelhetők meg, hanem az őket összekötő controller szintjén is. A beérkező utasítások alapján a controller hív tovább a megjelenítési és a model rétegbe.

FrameController

Alapvetőleg a vezérlő réteg objektumai között nincs egy egységes kapocs, amely összefogná őket. Az egyes vezérlő réteg objektumok nem ismernek más vezérlő rétegbeli objektumot, továbbá aktuálisan mindig pontosan egy controller van betöltve, így párhuzamosan sem futhatnak. Ez alól néhány controller kivételt képez. Ugyanakkor bizonyos esetekben szükség van más megjelenítési és vezérlési réteg betöltésére az éppen aktuálisan betöltött controller által. A `FrameController` eggyel magasabb absztrakciós szinten helyezkedik el, így oldva meg a fenti problémát. Amikor egy új

vezérlő rétegbeli objektumot töltünk be, akkor az objektumnak át kell adni egy **FrameController** objektum példányt, mint amolyan delegált objektumot, amelynek segítségével az egyes controllerek ki tudják cserélni a megjelenítési és vezérlési réteget. Fontos megjegyezni, hogy a csere a model réteget nem érinti.

CommunicatorController

A szervernek értesíteni kell a klienseket az adott asztalnál történő eseményekről. A kliensek ennek elérése végett egy publikus interfész biztosítanak és feliratkoznak a szerverhez. A szerver a kliensek beregisztrált objektumait nyilván tartja, és ezeken keresztül tudja értesíteni őket. A kliensek erre a feladatra egy külön objektumot - **CommunicatorController** - hoznak létre, és küldenek el a szerverhez. Az üzenetváltó objektumok az éppen betöltött kliens oldali vezérlő rétegbeli objektumot delegált objektumként konstruktori paraméterként kapják meg, így a szerver által küldött üzenetek fogadása után az üzeneteket tovább tudják küldeni a vezérlő rétegnek.

PokerRemoteObserver

Minden olyan kliens oldali vezérlői rétegbeli objektumnak implementálnia kell ezt az interfészt, amely működése során üzenetet fogad a szervertől, amelyet továbbít a kliens további rétegeinek.

12.5. Szerver modul

22. ábra. A szerver publikus interfésze



A szerver feladata a játék biztosítása a kliensek számára. A 22. ábrán látható a szerver jóldefiniált publikus interfésze. A kliensek ezt az interfészt tudják elérni az RMI registry-ből. A `PokerRemote` interfészt a `PokerRemoteImpl` osztály implementálja, ennek egy instanciája a tényleges szerver objektum. A szerveren futnak a játékasztal-szerverek, amelyek egy-egy asztalnak feleltethetőek meg, amelyekhez a kliensek csatlakozni tudnak. Ha egy játékos csatlakozik egy játékasztalhoz, akkor az adott asztal feljegyzi a csatlakozott klienst egy lokális listába. A főszervernek továbbra is van referenciaja a klienshez, ugyanis minden klienstől érkező utasítást a főszerver fogad, és irányít a megfelelő játékasztal-szerverhez. Erre az autentikáció miatt van szükség. Azonban ez a fajta megközelítés könnyen szűk keresztmetszetté válhat [13].

A szerver a jelszavak titkosítására bcrypt [14] eljárást alkalmaz, amelynek a biztonságát sózással növeli. Továbbá a szerver felhasznál még egy külső csomagot - mysql-connector-java -, amely az adatbázis kapcsolatért felel.

12.6. `javapokertexasholdem` modul

Külső könyvtár, amely eredetileg Texas Hold’Em játékstílusban tud kezeket kiértékelni. Az integrációt viszonylag könnyen meg lehetett oldani, ráadásul még általánosítani is sikerült a könyvtárat. Tehát classic és Texas Hold’Em játékstílusra is ugyanezt a könyvtárat használja a szerver a kezek kiértékelésére.

13. Funkciók

A szerver vázáért a `PokerRemote` interfész felel, amely a játékon végrehajtható műveletek összefogásáért felel. Itt található az összes funkció, amely megvalósításra került, mint például játék asztal létrehozása, új felhasználó létrehozás, admin jog kiosztása stb. A kliens ezt a vázat tudja elérni az RMI registry-ből, mint kliens-oldali szervercsonk, amelyeken a műveletek meg tudja hívni.

13.1. Általános funkciók

Regisztráció

A játékot csak regisztrált felhasználók tudják használni. A szerver biztosít regisztrációs lehetőséget a felhasználók számára. A regisztráció során felhasználónévét és jelszót köteles megadni a felhasználó, amely tárolásra kerül az adatbázisban. minden újonnan regisztrált felhasználó 1000 egységenyi egyenleggel rendelkezik. Az adatbázis séma nem engedi, hogy több azonos felhasználónévvel különböző felhasználók regisztrálva legyenek, erre a regisztráció során ügyelnie kell a felhasználónak.

Bejelentkezés

A játék használatához a játékosoknak be kell jelentkezniük. Egy felhasználónével egyidőben csak egy munkamenet létezhet. Ha a felhasználó már használatban lévő felhasználónével kíván beljelentkezni, akkor a szerver hibás bejelentkezési adatokra hivatkozva megtagadja az utasítást. Sikeres bejelentkeést követően a szerver példányosít egy PokerSession objektumot egy UUID [19] azonosítóval, amelyet lokálisan eltárol, majd a kliensnek elküld. A kliens a kijelentkezésig megőrzi ezt az objektumot, és a későbbi funkciók hívása során ezzel a munkamenettel azonosítja magát.

Kijelentkezés

A játékosok a kijelentkezés funkcióval hagyhatják el a játékot. A funkció hívása során a kijelentkezni kívánó felhasználó munkamenetét a szerver érvényteleníti, és a felhasználó visszakerül a bejelentkezési formhoz. A szerver (korlátozott mértékben) fel lett készítve a nem szabályszerű kijelentkezésre, tehát ha például megszakad a kapcsolat a klienssel, akkor a legközelebbi bejelentkezési funkció hívása során a szerver érvényteleníti a megszakadt kliensek munkameneteit, ezzel elérve azt, hogy a foglalt felhasználónevekkel újra be lehessen jelentkezni.

Asztalok

Létrehozás

A funkció eléréséhez admin jogosultságra van szükség. Egy új asztalt lehet létrehozni az adatbázis séma (hivatkozás) irányelvei szerint megfelelően paraméterezve. Amikor a kliens asztalt szeretne létrehozni, akkor a szerver egyértelműen tájékoztatja, hogy az asztal létrehozás sikeres volt-e. Ha igen, akkor egy felugró üzenet jelenik meg a képernyőn, hogy az asztal létrehozása sikeres volt, majd a felhasználó vissza lesz irányítva a tábla listázó felületre. Ha az asztal létrehozása sikertelen volt, akkor a felület ugyancsak egyértelműen jelzi a felhasználónak, hogy a művelet miért hiúsult meg. Például előfordulhat hibás beviteli érték, szerverhiba és adatbázishiba is.

Módosítás

Egy már meglévő asztalt adminisztrációs joggal rendelkező felhasználó módosíthat. Fontos megjegyezni, hogy a módosítás csak üres asztalokra alkalmazható.

Törlés

A funkcióhoz ismételten adminisztrációs jog szükséges. Már létező asztal törölhető a funkció segítségével. A funkciót csak üres asztalokra lehet alkalmazni.

Lekérés

A bejelentkezést követően a felhasználó a táblalistázó felületre kerül, ahol megtekint-heti a létrehozott és elindított játéktábla-szervereket (hivatkozás). A bejelentkezést követően a kliens ezeket az asztalokat elkéri a szervertől. Fontos megjegyezni, hogy ha bármelyik asztal módosításra került, akkor az asztalokat a szerver újra leküldi a klienseknek, így minden kliensnél az asztalok legfrissebb változata lesz elérhető.

Csatlakozás

A felhasználók létrehozott játékasztal-szerverekhez csatlakozhatnak. Fontos megje-geyezni, hogy egy asztalnál maximum 5 játékos ülhet egyszerre, tehát ha egy asztal tele van (5 játékos ül az asztalnál, beleszámolva a következő partira váró játékoso-kat is), akkor kliens oldali kód nem engedi beülni a felhasználót az adott asztalhoz, amelyről a felhasználót a képernyőn értesíti. Ha egy tele asztaltól kilép az egyik játékos (vagy megszakad a kommunikáció), akkor az asztalnál felszabadul egy hely, amelyet egy új felhasználó elfoglalhat, ekkor az asztal ismét tele lesz.

Szabad helyek lekérdezése

Ha egy kliens be szeretne ülni az egyik játékasztal-szerverhez, akkor előbb le kell kérdeznie, hogy van-e még szabad hely az asztalnál.

Felhasználók

Létrehozás

A regisztráció során új felhasználó entitás jön létre, amelyet a szerver letárol az adatbázisban. A funkció külön, direktbe nem hívható, regisztrációhoz kötött.

Egyenleg visszaállítás

A felhasználók egyenlegét adminisztrációs joggal rendelkező felhasználó vissza tudja állítani alaphelyzetbe, amely 1000 egységnek felel meg.

Törlés

Ugyancsak adminisztrációs jogot követel meg egy felhasználó végleges eltávolítása a póker játékból.

Lekérés

Az adminisztrációs joggal rendelkező felhasználók lekérdezhetik az adatbázisból az összes regisztrált felhasználót. A felhasználók a képernyőn táblázatos formában je-lennek meg, amelyből felhasználót ki lehet választani a további funkciók eléréséhez.

Jelszó csere

A felhasználóknak lehetőségeük van jelszócserére.

Adminisztrációs jog

A szerverre beregisztrált felhasználók alapesetben nem rendelkeznek ilyen jogosultsággal. Az adminisztrációs jogot csak adminisztrációs joggal rendelkező felhasználó tudja kiosztani illetve megvonnai. Friss adatbázis esetén létezik egy admin nevű felhasználó, aki adminisztrációs joggal rendelkezik. Kezdetben csak ez a felhasználó tud kiosztani adminisztrációs jogot.

13.2. Játékmenetbeli funkciók

Utasítás küldés

A szerver és a kliens kommunikációja a programcsomag által definiált üzenetobjektumokkal történik. Bármilyen játékközbeni utasítást szeretne küldeni a kliens a szervernek, vagy éppenséggel a szerver a kliensnek, azt be kell csomagolnia egy **PokerCommand** interfész implementáló osztályba. A szerver és a kliens parti közben csak ilyen típusú üzenetek fogadására alkalmas. Pontosabban fogalmazva az implementációs nehézségeken túllendülve a klienseket fel kellett készíteni arra, hogy a szerver adott esetben megpingelheti őket.

Egyenleg lekérése

A szerver az automatikus egyenleg frissítést nem támogatja. Az egyes klienseknek egyenként kell arról gondoskodniuk, hogy a grafikus felületen minden az éppen aktuális egyenlege jelenjen meg a játékos számára.

14. Tesztelés

14.1. Funkcionális tesztelés

Funkció	Elvárt eredmény	Eredmény
Regisztráció	A felhaszáló a regisztráció gomb megnyomása után tájékoztatást kapjon afelől, hogy a regisztráció sikeresen megtörtént.	A program az elvártnak megfelelően működött.
Bejelentkezés	A bejelentkezési formot helyesen kitöltve és a bejelentkezés gombra kattintva a program sikeresen autentikálta és átirányította a felhasználót a táblalistázó felületre.	A program az elvártnak megfelelően működött
Játék asztalhoz való beülés egyedüli játékosként	Az asztal betöltődik és a felhasználó profilján kívül a grafikus interfészen más profilkép nem jelenik meg. Tetszőleges idő eltelte után sem indulhat el a játék, kivétel ha legalább egy másik felhasználó csatlakozik az adott játék asztalhoz	A program az elvártnak megfelelően működött.
Játék asztalhoz való beülés nem egyedüli játékosként	Ha az asztalnál éppen parti zajlik, akkor ebből az újonnan csatlakozott játékos semmit nem érzékel. A játékos a következő partiba szállhat be. A következő parti megkezdődik, ha legalább kettő darab játékos ül az asztalnál.	A program az elvártnak megfelelően működött.
Tábla módosítás	-	-
Tábla törlés	-	-

15. Tovább fejlesztési lehetőségek

- Az adatbázis viszonylag alacsony absztrakciós szinten került implementására, azonban mivel néhány tábláról beszélhetünk csak, ezért igyekeztem elkerülni a keretrendszerek általi overheadet. Ugyanakkor ezen a ponton sokat fejlődhet a programcsomag, ha a későbbiek során esetlegesen bonyolultabban kellene modellezni a játékot adatbázis szempontjából. Például dialektusok - akár Liquidbase (hivatkozás) - használata elfedheti a tényleges adatbázis-kezelő rendszer általánosságait, így eggyel magasabb szintre helyezhető a megvalósítás.
- A felhasználói élményen sokat javíthat az animációk használata. A megjelenítés sokkal lágyabb, folyékonyabb lehetne Transition/Animation (bibliográfiába hivatkozás...) objektumok használatával.
- Akár a komplet RMI architektúrát le lehetne váltani, és helyette REST szoftverarchitektúrát alkalmazni, amely modernebb megjelenést (AngularJS, re-szponzív design) és modernebb fejlesztői eszközöket, API-kat vonna maga után.
- A játék nem képes kezelni olyan eseteket, amikor egynél több játékos nyer az adott körben.
- A játék nincs felkészítve arra a szélsőséges esetre, ha valakinek elfogy a zsetonja, akkor pontosan minek (és hogyan) kell történnie.
- A kódban viszonylag sok kód duplikáció van jelen, ugyanis az HouseCommandType és a PlayerCommandType enum típusú objektumok szűk keresztmetszetnek tudható be. Ha a PokerCommand interfacet implementáló osztályokat generikusan tudnánk megfogalmazni, akkor jelentősen letisztulna a kód.
- Az admin jogot el lehetne távolítani a játékból, és helyette MVC tervezési minta alapján a szerver oldalra is implementálni lehetne egy grafikus interfész, amelyen keresztül a szerver teljeskörű karbantartása és adminisztrációja elvégezhető lenne.
- A programcsomag a váratlan adatbázis hibákat nem tudja megfelelően lekezelni. Ezek javításra szorulnak.
- A programcsomag szerver oldali kódja a konkurens helyzeteket nem mindenig kezeli le helyesen. Például ha egy játékasztalhoz a kliensek gyorsan fellecsatlakoznak. Ebből fakadóan előfordulnak bugok és szükség volt kliens oldalon és szerver oldalon is egy-egy mentőötvet implementálni. Kliens oldalon a

Q betű segíti a felhasználókat a hibás játékasztal elhagyásában, szerver oldalon pedig egy reset nevű metódus került implementálásra, amit adminisztrációs joggal lehet meghívni.

IV. rész

Irodalomjegyzék

16. Hivatkozások

- [1] Java SE Runtime Environment 8 letöltése
<http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html>
- [2] MySQL Community Server 5.6 letöltése
<https://dev.mysql.com/downloads/mysql/5.6.html>
- [3] Póker játék
<https://hu.wikipedia.org/wiki/P%C3%A1sker>
- [4] OMAHA póker
https://hu.wikipedia.org/wiki/Omaha_p%C3%A1sker
- [5] Kártyakombinációk
[https://hu.wikipedia.org/wiki/K%C3%A1rtyakombin%C3%A1ci%C3%A1sk_\(p%C3%A1sker\)](https://hu.wikipedia.org/wiki/K%C3%A1rtyakombin%C3%A1ci%C3%A1sk_(p%C3%A1sker))
- [6] Ötlapos póker
https://hu.wikipedia.org/wiki/%C3%89tla pos_p%C3%A1sker
- [7] Texas Hold'Em
https://hu.wikipedia.org/wiki/Texas_Hold'Em
- [8] Eclipse Mars IDE
<https://projects.eclipse.org/releases/mars>
- [9] Eclipse Quick Search plugin
<http://spring.io/blog/2013/07/11/eclipse-quick-search>
- [10] Java Poker Texas Holdem Hand Evaluator
<https://github.com/phstc/javapokertexasholdem>
- [11] Plain Old Java Object
https://hu.wikipedia.org/wiki/Plain_Old_Java_Object
- [12] Object-relational mapping
https://hu.wikipedia.org/wiki/Objektum-rel%C3%A1ci%C3%A1ss_lek%C3%A1lpez%C3%A1ls
- [13] UUID
<https://en.wikipedia.org/wiki/Bottleneck>

[14] bcrypt

<https://en.wikipedia.org/wiki/Bcrypt>

[15] Java RMI

https://hu.wikipedia.org/wiki/Java_remote_method_invocation

[16] Observer pattern

https://en.wikipedia.org/wiki/Observer_pattern

[17] Model-View-Controller

<https://en.wikipedia.org/wiki/Model-View-controller>

[18] JavaFX 2

<http://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm>

[19] UUID

https://en.wikipedia.org/wiki/Universally_unique_identifier