Önálló laboratórium – Dokumentáció

Android alapú szoftverfejlesztés Java nyelven

(2018/2019 2. félév)

Football Contest Creator

Készítette: Tóth László

Konzulens: Pomázi Krisztián

1. Bevezetés

Napjainkban egyre jobban elterjedtek az Android operációs rendszerrel rendelkező mobiltelefonok, tabletek, és egyéb okoseszközök. A Google Play Áruház tele van különböző applikációkkal, játékokkal, fájlkezelőkkel, zene és videólejátszó programokkal, a mindennapjainkat segítő, általunk adminisztrálható dolgokat kezelő alkalmazásokkal. Az általam elkészített applikáció az utolsó kategóriába tartozik.

A minden évben megjelenő FIFA az egyik leghíresebb sport kategóriájú videójáték, amely lehetőséget ad arra, hogy akár nagyobb baráti társaságok is megmérkőzhessenek egymással, hogy kiderüljön, ki a legjobb játékos. Futball tornák szervezése mindig is nagy népszerűségnek örvendett akár a szabadban, akár teremben a profi és az amatőr csapatok között is, legyen ez a torna egy komplett bajnokság vagy egy egyenes kieséses kupasorozat.

Bármilyen eseményről is legyen szó, a meccsek eredményeinek és a csapatok, játékosok statisztikáinak adminisztrálása és nyilvántartása elengedhetetlen. Ennek megkönnyítésének érdekében hoztam létre a **Football Contest Creator** nevű Androidos applikációt.

2. Specifikáció

Az applikáció felhasználója létrehozhat különböző csapatokat és tornákat, amelyeknek nevet adhat, a tornáknak meghatározhatja a típusát, a körök számát, a résztvevő csapatok számát, valamint a verseny típusát, amelyet a következők közül választhatja ki:

- **Bajnokság:**

  Fordulókra osztott torna, amelyben mindenki mindenkivel játszik, ami alapján kialakul egy végső sorrend a csapatok között. Az eredményeket az alkalmazás egy tabellán rögzíti, amelyen látható a csapatok helyezése, az elért pontszámok, a győzelmek, döntetlenek, vereségek száma, valamint az összesített gólkülönbségek. Az értelmezést színkódok is segítik.

- **Egyenes kieséses:**

  Szakaszokra osztott torna, amelynél minden szakaszban az alkalmazás összepárosítja a csapatokat, és csak a győztes jut tovább a következő szakaszba. Az eredményeket az alkalmazás egy ágrajzon rögzíti.

Körök tekintetében a felhasználó az alábbi opciók közül választhat (annak figyelembevételével, hogy egy kör egyenlő egy mérkőzéssel két tetszőlegesen választott csapat között):

- **Egy meccses**
- **Oda-visszavágós**
- **Több meccses** (csak bajnokságnál elérhető)

Ezek alapján egy egyenes kieséses torna esetében a körökkel megegyező számú mérkőzésen dől el a továbbjutás sorsa, bajnokság esetében pedig minden csapat a körökkel megegyező számú mérkőzésen találkozik minden másik csapattal.

A csapatok és a tornák a létrehozás után egy listába kerülnek, ahol a felhasználó megjelölheti a kedvenceit az adott kategóriában. A kedvencek a listák elején foglalnak helyet. Ezek után egy elemre kattintva megjelenik annak részletezése.

Csapatoknál egy kördiagramon láthatók az eddig elért eredmények. A felhasználó további füleken tekintheti meg az adott csapat meccseinek eredményeit, hogy milyen meccsek folynak éppen, vagy lesznek a jövőben, valamint a csapathoz felvett megjegyzéseket, információkat, amelyeket szerkeszteni is tud.

Tornáknál a tabella vagy az ágrajz mutatja az aktuális állást. A felhasználó itt is több fülön találhatja meg az adott tornához tartozó összes meccset, legyen az

végeredmény, vagy csak részeredmény. Továbbá itt is van lehetőség megjegyzéseket hagyni az adott tornával kapcsolatban.

A felhasználó több ponton is beviheti az egyes meccsekhez tartozó eredményeket, amelyeket szükség esetén módosíthat is, egészen az eredmény véglegesítéséig. Ezen eredmények alapján az alkalmazás folyamatosan frissíti a statisztikákat, hogy azok mindig naprakészek legyenek.

A különböző adatok (tornák, csapatok, meccsek, eredmények, statisztikák stb.) adatbázisban tárolódnak, amelynek tartalma egyszerűen módosítható, és könnyen olvasható a program számára.

3. Felhasznált technológiák

Az alkalmazás elkészítését megelőzően felmértem a jelenleg forgalomban lévő Androidos eszközök verzióbeli eloszlását, és megállapítottam, hogy 15-ös API (Application Programming Interface) szintig visszamenőleg le lehet fedni azok alig kevesebb, mint 100 százalékát. Így ezt választottam minimálisan támogatott verziónak.

Az applikáció teszteléséhez a saját mobiltelefonomat választottam, amelyen 26-os szintű API van. Így a programot egy valós eszközön próbálhattam ki, amely a legjobb választás, ha a környezeti tényezők hatásait, illetve az azokra való ráhatást is figyelembe kell venni a fejlesztés során.

Adatbázis-kezeléshez a Room Persistence Library-t használtam, amelyet a Google fejlesztett ki. Ez egy absztrakciós réteget szolgáltat az SQLite adatbázis-kezelő rendszer felett. Segítségével könnyen lehet adatbázis műveleteket elvégezni egyszerű annotációk felhasználásával.

Az alkalmazás teszteléséhez fontos volt tudnom, hogy az adatbázisom helyesen tudja-e kezelni mindazon adatokat, amiket a felhasználó el akar tárolni benne. Ehhez felhasználtam egy olyan könyvtárat (Android Debug Database), amivel fordítás után böngészőből tudtam figyelemmel kísérni az adatbázis tartalmát. Ennek további előnye volt, hogy azonos hálózatra csatlakoztatva a laptopot és a mobiltelefont, amiről az applikációt indítottam, képes voltam vezeték nélküli kapcsolattal is ellenőrizni az adatbázis helyességét.

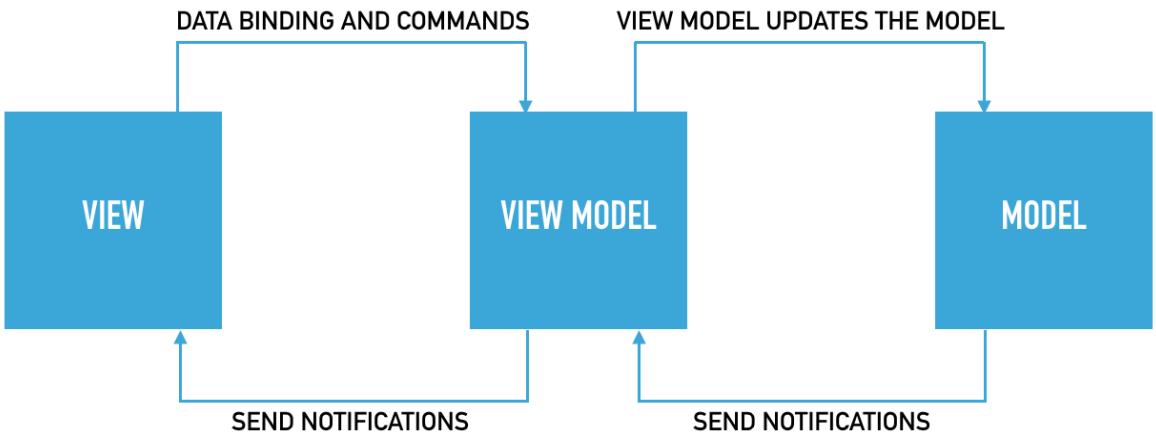
Az alkalmazásban többször használt LiveData osztály egy megfigyelhető objektumot biztosít a listák (tornák, csapatok, meccsek, tabellák) folyamatos

aktualizálásához. Továbbá figyeli az alkalmazás komponenseinek életciklusát, és csak az aktív komponenseket frissíti, amivel erőforrásokat takarít meg.

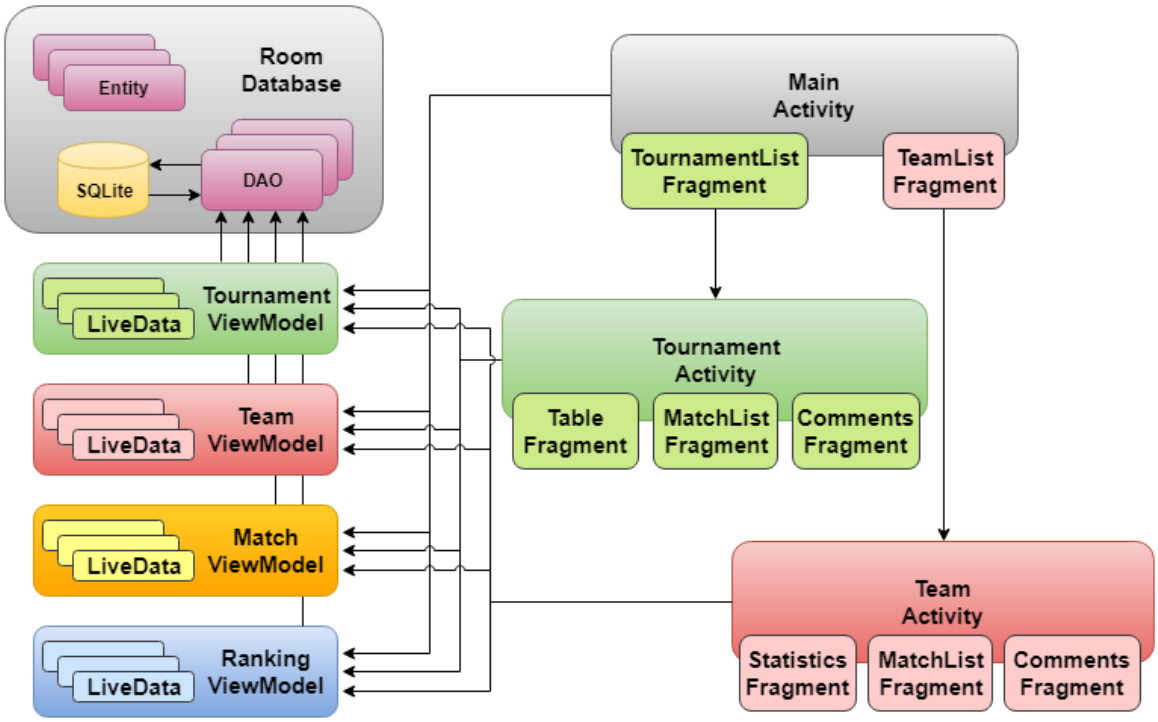
A csapatok statisztikájának kimutatásához használt kördiagramot is egy külső könyvtár felhasználásával valósítottam meg (MPAndroidChart). Ezzel egyszerűen, csupán a statisztikai adatok megadásával meg lehet jeleníteni a kívánt diagramot.

4. Tervezés

Az applikáció megvalósítását az architektúra megtervezésével kezdtem. Ennek során az MVVM minta (Model - View - ViewModel) elérésére törekedtem, ez egy igen gyakori architekturális forma, amelyben jól elkülönülnek a fenti részek. Ezáltal jól definiált interfészeken keresztül tudunk kommunikálni az üzleti logika (Model), a megjelenítés (View) és a megjelenítés logikája (ViewModel) között.



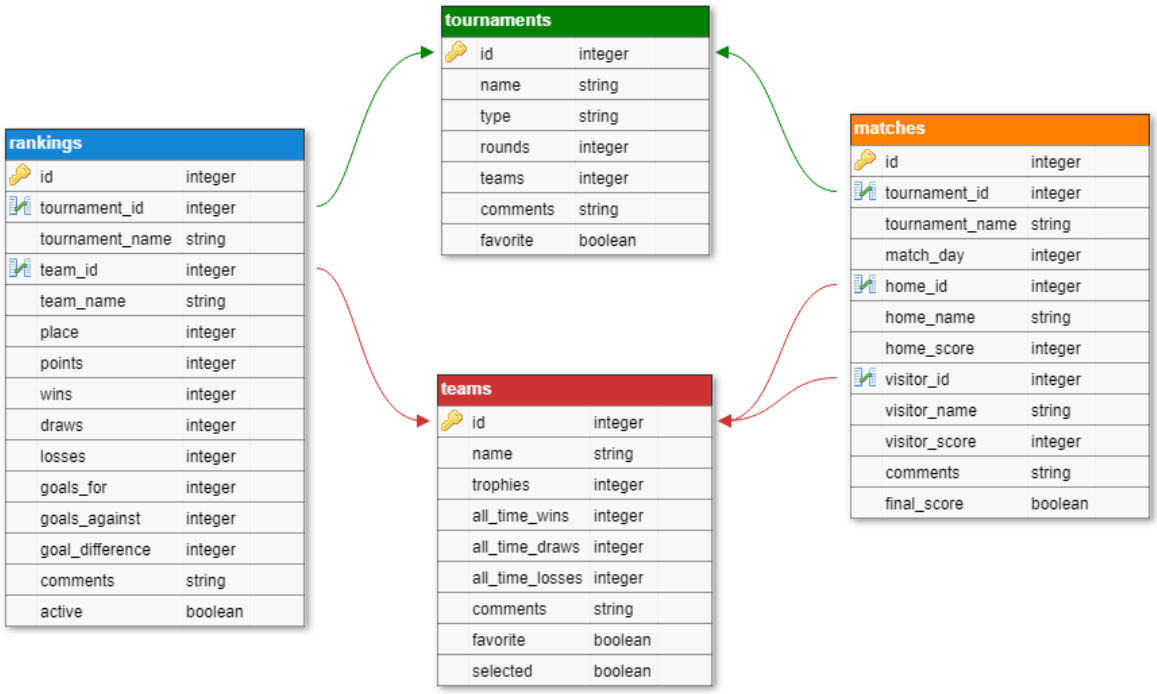
A projekt felépítésénél a fenti mintát figyelembe véve az alábbi programszerkezetet valósítottam meg.



4.1. Adatbázis séma

Az adatbázis sémájának megtervezésével kezdtem a munkát. Ezt időközben többször kellett módosítani, hogy ki tudja elégíteni azokat az igényeket, amelyek később merültek fel. Ilyenkor új attribútumok felvétele, valamint a már meglévők típusának megváltoztatása jelentette a megoldást. Ezenkívül bizonyos változók kezdeti értékének módosítása is szükséges volt a helyes működés érdekében.

Az applikáció adatbázisa négy táblából áll. Mindegyik megalkotásánál azokat az attribútumokat vettem figyelembe, amelyek jól használhatók az alkalmazásban. Így minden entitáshoz feltétlenül szükséges egy egyedi azonosító, amellyel garantálható a megkülönböztethetőség. Ez az egyediség a Room segítségével könnyen kivitelezhető, ugyanis lehetőség van automatikus generálás beállítására (autoGenerate), amely minden egyes újonnan felvett egyedhez a soron következő azonosítót állítja be. Minden további tulajdonság táblánként eltérő, így ezeket az alábbi pontokban mutatom be.



4.1.1. „tournaments” tábla

A tornáknál szükség van annak nevére, amelyet az egyszerűség kedvéért egyedi attribútumként vettem fel, így a **MainActivity**-ben található listában nem található két ugyanolyan elem. Kötelező még megadni a torna típusát, amely lehet „Bajnokság” vagy „Egyenes kieséses”, valamint a körök és a résztvevő csapatok számát. Ezek segítenek a későbbiekben az applikáció leglényegesebb részében, a meccsek megalkotásában. Ezenkívül eltároljuk még a tornához, hogy a felhasználó megjelölte-e kedvencként, hogy a tornalista elején foglaljon helyet, valamint a hozzá tartozó megjegyzéseket, például a dátumot, helyszínt és minden egyéb, az applikáció által automatikusan nem dokumentált információt.

4.1.2. „teams” tábla

A csapatok létrehozásakor is szükséges megadni annak nevét, a többi attribútumot alapértelmezetten állítja be az applikáció. Ezek a megszerzett trófeák száma és a későbbi statisztikák alapját képező győzelmek, döntetlenek és vereségek száma. Ezenkívül a tornáknál látott módon lehet eltárolni az adatbázisban a megjegyzéseket (csapattagok, sérülések, edző stb.), a kedvenc státuszt, valamint a tornák létrehozásánál szükséges kiválasztott státuszt, amellyel azt lehet megjelölni, hogy mely csapatok vesznek részt az adott tornán.

4.1.3. „matches” tábla

A meccseknél több, a tornák és csapatok táblából származó információra is szükség van, ezek az azonosítók és a nevek. Minden meccs egy bizonyos tornához tartozik és két csapat között jön létre, így azok attribútumait külön-külön fel kell vinni az adatbázisba. Továbbá a több, mint két körből álló bajnokságok esetében nem lenne egyértelmű, hogy hányadik körben zajlik egy adott meccs, így el kell tárolni a meccsnapot is. A már folyamatban lévő és lezárt meccseknél szükség van az eredményekre, ezért a két csapat lőtt góljainak száma is tárolásra kerül. A megjegyzések ennél a táblánál sem maradhatnak el, például a sárga-, és piroslapok, gólszerzők stb. Végül pedig szükség van egy a végleges eredményt mutató attribútumra is, amely a későbbiekben segít a meccsek rendszerezésében.

4.1.4. „rankings” tábla

Minden torna minden csapatához tartozik egy helyezés, amely a tabellán, vagy ágrajzon elfoglalt helyet és a tornán elért eredményeket mutatja. A helyezések táblánál van szükség a legtöbb attribútumra. A torna és a csapat azonosítóján és nevén kívül az aktuális helyezés, az elért pontok, a győzelmek, döntetlenek és vereségek száma, amelyekből a megszerzett pontokat számítja az applikáció, valamint a lőtt gólok, a kapott gólok és ezek aránya mind fontos információk a tornán elért helyezés szempontjából. Itt is van lehetőség megjegyzések elhelyezésére, például a pontbeli büntetéseket, amelyeket később manuálisan tervezünk levonni az elért pontszámból stb. Ezenkívül egy további attribútum szolgál arra, hogy jelezze az egyenes kieséses tornák során a még bent lévő és a már kiesett csapatokat, mellyel a megjelenítést és az új körök sorsolását segíti elő.

4.2. Adatbázis lekérdezések

Az adatbázis lekérdezések a Room-ban DAO-kon (Data Access Object) keresztül valósulnak meg. Ezek interfészek az adatbázis és a program többi része között. Itt definiáltam azokat az SQL-lekérdezéseket, amelyekkel az adatbázis tartalmát szűrtem bizonyos funkciók eléréséhez, de a Room segítségével beépített műveleteket is végre tudtam hajtani, ilyenek a beszúrás (Insert), módosítás (Update) és törlés (Delete). Ezek közül az első kettőhöz be lehetett állítani, hogy a művelet végrehajtása közben történő konfliktust hogyan kezelje. Itt az Ignore, azaz figyelmen

kívül hagyást választottam, mivel nem volt szükséges, hogy az applikáció kezelje az ily módon felmerülő problémákat, ugyanis ezeket az eseteket a program többi részénél megfelelően kezeltem. Ennél az interfésznél jelenik meg először a már korábban említett LiveData, bizonyos lekérdezések eredményét ebbe az objektumba csomagolva használtam fel a későbbiekben.

4.3. ViewModel-ek

A ViewModel feladata, hogy egyfajta hídként szolgáljon az üzleti logika és a megjelenítő között. Ez esetemben azt jelenti, hogy továbbítja az adatokat az adatbázisból az alkalmazás Activity-jeibe, és Fragment-jeibe, valamint frissíti az adatbázis tartalmát a felhasználói felületen végzett műveletek szerint. Adatok megosztására is használható több Fragment között. Továbbá fontos tulajdonsága, hogy túléli a konfigurációs változásokat, az Activity-k és Fragment-ek életciklusbeli változásait. Ez például olyan eseteknél hasznos, amikor a programot futtató készülék képernyője elfordul. Ilyenkor az Activity újraindul, és az addig használt értékek elvesznek, viszont ViewModel segítségével ez a veszteség elkerülhető.

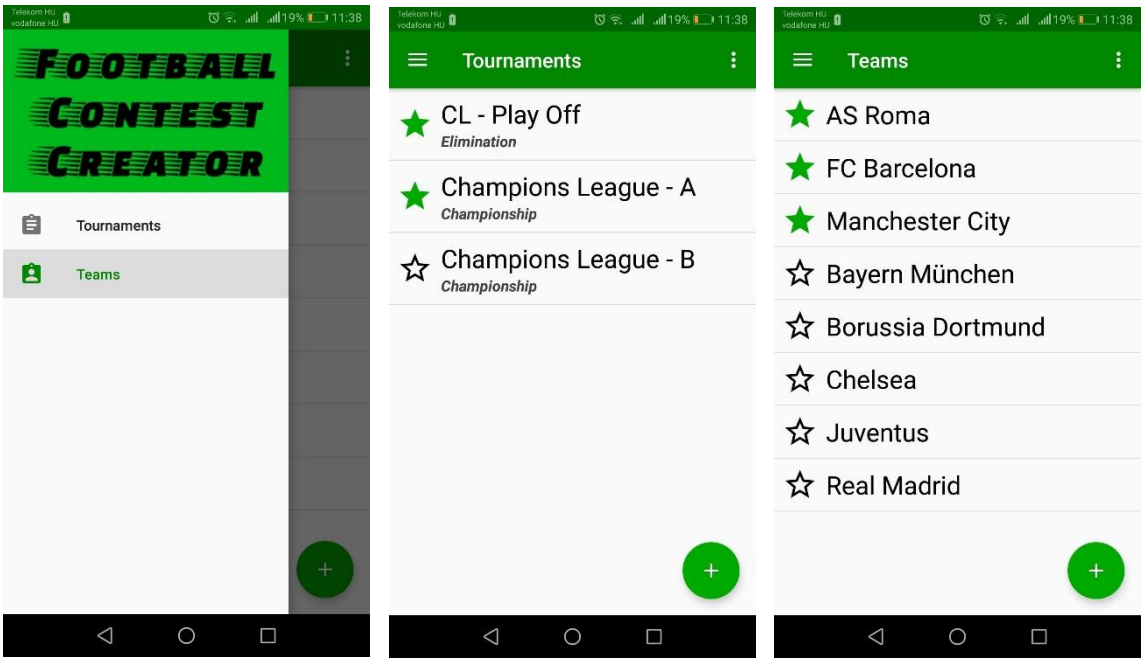
Az alkalmazásomban minden adatbázis táblához létrehoztam egy ViewModel-t, amin keresztül a DAO-kban definiált műveleteket hívom meg, így nem közvetlenül a UI-t (User Interface) terhelem ezzel a feladattal.

4.4. Activity-k és Fragment-ek

Az applikációban három Activity-t hoztam létre, ezek a program fő részei, legfőbb feladatuk a felhasználóval való interakció. Az első ezek közül a **MainActivity**, az alkalmazás belépési pontja. A felülete több részből tevődik össze. Egy Navigation Drawer segítségével lehet navigálni annak menüpontjai között, amelyek a tornák listája, valamint a csapatok listája. Ezeknek a megjelenítéséhez egy-egy Fragment-et hoztam létre (**TournamentListFragment** és **TeamListFragment**), amelyek rácsatolódnak az Activity-re, majd a megfelelő ViewModel-t felhasználva beállítanak egy megfigyelőt (Observer) a lekért adathalmazra, hogy minden változásról azonnal értesüljenek, és azokat meg is jelenítsék a képernyőn.

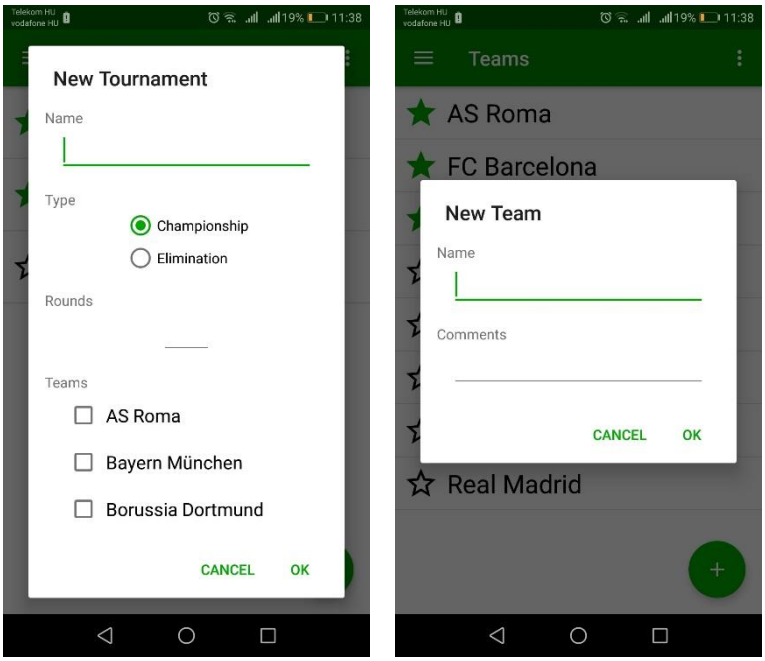
A listákat Recycler View-ban jelenítettem meg. Ez optimalizálás szempontjából is jó választás, mivel újrahasznosítja azokat az elemeket, amelyek aktuálisan nem a képernyőn helyezkednek el, ezáltal erőforrásokat takarít meg. A Recycler View-khoz

létrehoztam egy-egy adaptert is (**TeamListAdapter** és **TournamentListAdapter**), amelyekben meghatároztam, hogy egy listaelem mit jelenítsen meg, valamint, hogy milyen felhasználói interakciókra reagáljon. Megjelenítés szempontjából relevánsak az adott torna vagy csapat neve és kedvenc státusza, a tornáknál azok típusa is. Igénybe vett felhasználói interakciók a kattintás, amely az adott elem részleteit jeleníti meg egy teljesen új Activity-ben, a hosszú kattintás, amelynek hatására az adott listaelem kitörölhető az adatbázisból, valamint a kedvencet jelölő csillag szimbólumra kattintás, amellyel a felhasználó megjelölheti a számára fontos elemeket. Az adapterek tartalmazzák magukat a listákat is, amelyeket a Fragment-ben elhelyezett megfigyelő frissít minden változás alkalmával. Mivel az adatbázisból rendezett listákat kértem le ezeknek a listáknak a megjelenítéséhez, így látványosan minden kedvenc gombra kattintáskor az adott elem a lista elejére ugrik, és ez a LiveData-nak köszönhető. Az adapterekben lévő listener-eket implementáltam a Fragment-ekben, így már ténylegesen is interakcióba lehet lépni a listaelemekkel.



A Fragment-ek alján látható Floating Action Button-ök szolgálnak új tornák, illetve csapatok felvételére. Ezek megnyomására előugranak a Dialog Fragment-ek (**NewTournamentDialogFragment** és **NewTeamDialogFragment**), amelyeken egy rövid kérdőív kitöltése után létrejönnek a már előzőekben említett listák új elemei, amelyek az adatbázisban is rögzítésre kerülnek. Az új tornák felvételéhez használt Dialog Fragment-ben szükség volt egy újabb Recycler View felvételére ahhoz, hogy a

résztvevő csapatokat ki lehessen választani. Ehhez egy új adaptert is létrehoztam (**TeamSelectionListAdapter**), amelyben a Checkbox-okra kattintással történő kijelölést figyeli az alkalmazás.

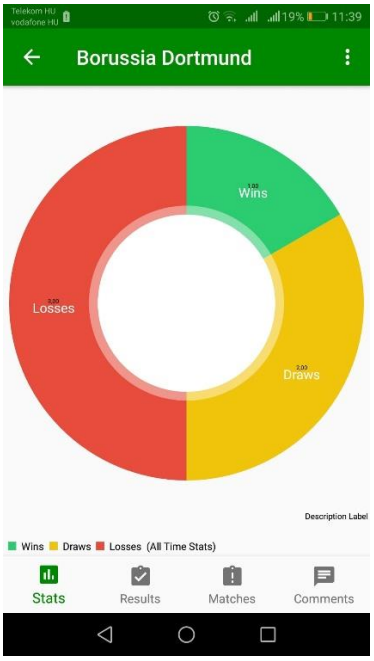


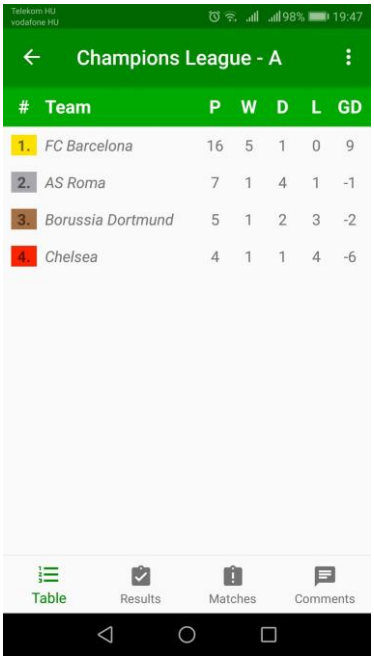
Bármely listaelemre történő kattintás hatására új Activity indul el, ahol megtekinthetők, valamint módosíthatók a statisztikák, eredmények és megjegyzések. Ezen Activity-k (**TournamentActivity** és **TeamActivity**) Bottom Navigation View típusú menüt használnak, amelyeken négy elem található, ezek a Statisztika/Tabella, Eredmények, Meccsek és Megjegyzések névvel szerepelnek az applikációban, melyek mindegyikéhez külön Fragment tartozik (**StatisticsFragment**, **TableFragment**, **MatchListFragment** és **CommentsFragment**).

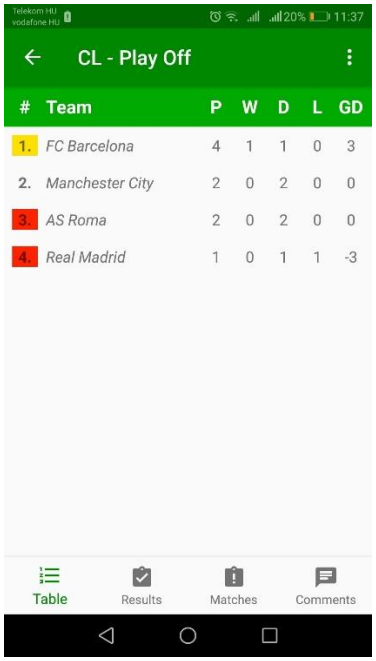
A **StatisticsFragment** a csapatok statisztikáinak megjelenítésére szolgál, egy kördiagram segítségével szemlélteti az applikáció a győzelmek, döntetlenek és vereségek arányát.

A **TableFragment** a tornák tabelláit jeleníti meg. Még nem valósult meg az a funkció, amivel egyenes kieséses tornáknál az ágrajzot jeleníti meg a program, így jelenleg ezt is tabella megjelenítésével oldottam meg, ahol másfajta színkombinációt használtam fel a kiesett csapatok, valamint a győztes csapat kiemeléséhez, mint a bajnokságoknál, ahol a dobogósokat és az utolsó helyen álló csapatot jelöli megkülönböztető szín. Ennél a Fragment-nél szintén szükség volt egy Recycler View implementálására, valamint a hozzá tartozó adapter (**RankingListAdapter**)

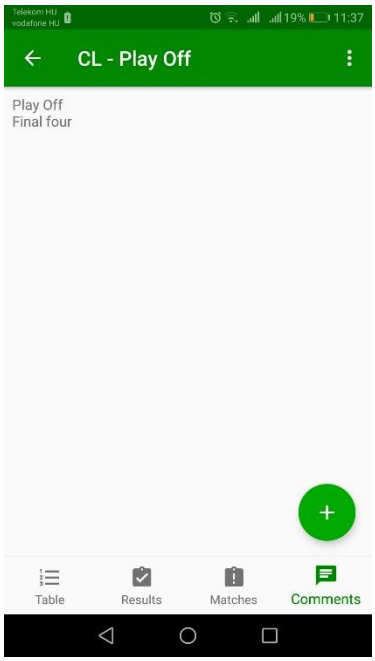
létrehozására. Egy elem tartalmazza az aktuális helyezést, a csapat nevét, pontszámát, győzelmek, döntetlenek, vereségek számát, valamint a gólarányt.



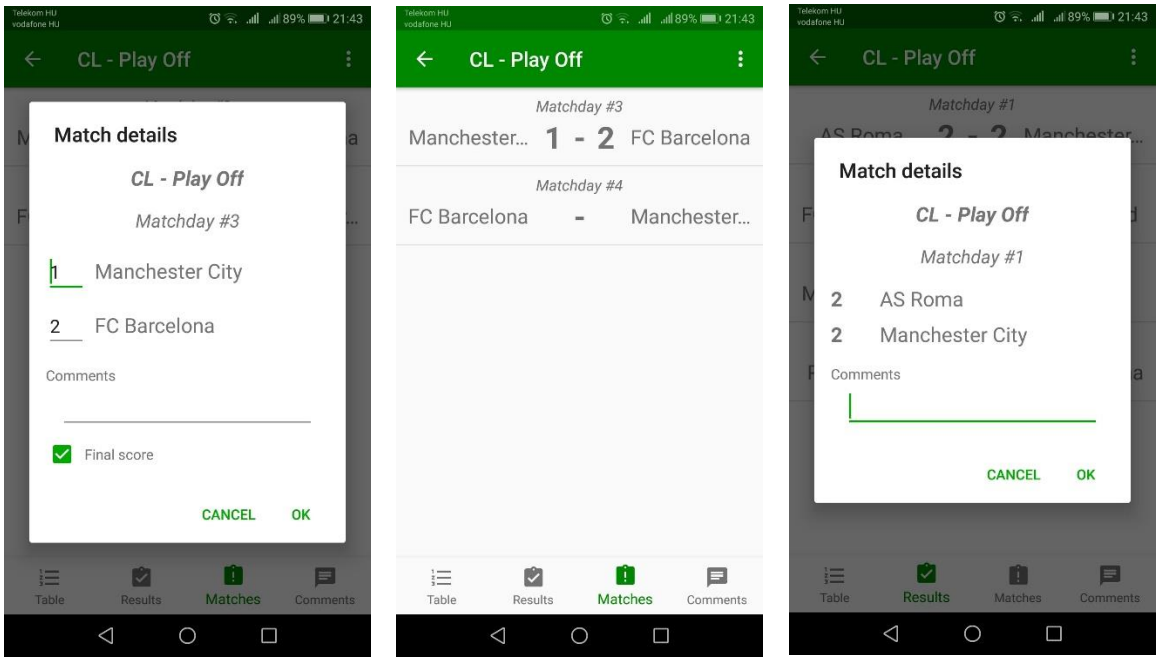




A megjegyzések felvételére szolgáló **CommentsFragment** felületén egy Floating Action Button segítségével lehet további kommenteket adni a tornákhoz és a csapatokhoz is (**CommentsDialogFragment**).



A **MatchListFragment** egy olyan Fragment, amelyet több helyen is hasznosítani tudtam a programomban. Attól függően, hogy egy adott tornához vagy csapathoz tartozó meccseket listáz, vagy hogy már lejátszott vagy folyamatban lévő meccsekről van szó, négy helyen is lehet találkozni az applikációban ezzel a felülettel. Mivel szintén egy listát megjelenítő Fragment-ről van szó, ennél is Recycler View-val, valamint az ahhoz tartozó adapterrel (**MatchListAdapter**) oldottam meg ezt a feladatot. A lista egy eleme megjeleníti a meccshez tartozó meccsnapot, a két csapat nevét és a szerzett gólok számát. A meccslista elemeire való kattintás hatására kétfajta Dialog Fragment jelenhet meg a képernyőn (**MatchDetailsDialogFragment** vagy **ResultDetailsDialogFragment**). Előbbinél tudja a felhasználó módosítani a meccsek eredményét és megjegyzéseket írni hozzá, valamint véglegesíteni az eredményt. Utóbbinál már csak a végeredmény megtekintésére van lehetőség, illetve további kommenteket fűzni a meccshez.



A legtöbb Activity-nél felhasználtam a Toast adta lehetőséget, és különböző értesítéseket állítottam be, amelyek megjelennek a felhasználónál, amint végrehajtott egy műveletet, és értesíti őt annak sikerességéről vagy az esetleges hibákról.

4.5. Torna generálásának logikája

Egy torna létrehozását sok tényező befolyásolja. Ezek a torna típusa, a körök száma, valamint a résztvevő csapatok száma.

Bajnokság esetében minden csapat minden másik csapattal játszik annyi meccset, amennyi a felhasználó által megállapított körök száma. Ebben az esetben a meccsalkotást azzal kezdtem, hogy minden csapathoz hozzárendeltem minden másik csapatot, és az azonosakat kiszűrtem. Így, ha **n** számú csapat vesz részt a tornán, akkor **(n * (n - 1)) / 2** számú meccs jön létre, amelyek egy kört tesznek ki. Ezután ezeket a meccseket meccsnapokra kellett beosztani azzal a megkötéssel, hogy egy meccsnapon egy csapat maximum egyszer játszhat. Az egy meccsnapon lejátszott meccsek száma függ a csapatok számától. Amennyiben ez páros, akkor **n / 2** meccs, ha páratlan, akkor **(n – 1) / 2** meccs zajlik egy napon. Továbbá, a meccsnapok száma is hasonlóan függ a csapatok számától. Ha ez a szám páros, **n - 1** meccsnapból áll egy kör, ha páratlan, akkor pedig azzal megegyező, **n** meccsnapon játszanak egymással a csapatok.

A meccsnapokra való beosztásnál sok listát kellett fenntartanom azon meccsek miatt is, amelyek nem kerülhetnek sorra egy adott meccsnapon, és amelyekkel már próbálkozott az algoritmus, de sikertelenül. Végezetül pedig megkevertem a hazai-, illetve vendégcsapatokat, valamint a meccsnapon belüli beosztást is, így mindig különböző sorsolás jöhet létre.

Egyenes kieséses torna esetén figyelembe kellett vennem a selejtezők lehetőségét. Ilyen helyzet akkor adódik, ha nem kettő valamely hatványával megegyező számú csapat vesz részt a tornán. Ilyenkor a program kiszámítja, hogy melyik az a legnagyobb szám, amely kettő hatványa és nem nagyobb a csapatok számánál, majd ezt kivonja a csapatok számából, és veszi annak kétszeresét. Ezután kiválaszt ezzel megegyező számú csapatot, majd összepárosítja őket. Egyenes kieséses tornánál egy kör egy meccsnapnak felel meg, így a bajnokságoknál alkalmazott bonyolult beosztási műveletre itt nincs szükség. Továbbá, az applikáció minden kör végén felajánlja a felhasználónak az új kör generálását, amely az előbb leírtakhoz hasonló módon történik, de a selejtezők után már könnyebb az algoritmus feladata, hiszen pontosan kettő hatványaival kell számolnia. Ez abból is következik, hogy az ágrajz egy bináris fához hasonlítható.

A körök száma innentől kezdve annyiban befolyásolja a torna létrehozását, hogy az eddig megismert műveletek megismétlésére már nincs szükség. Mindössze annyi a feladat, hogy meg kell cserélni a hazai-, és a vendégcsapatokat. Továbbá annyi megkötés van a körök számát illetően, hogy amíg bajnokságoknál ez a szám korlátlan, addig egyenes kieséses tornáknál maximum két kör fogadható el.

4.6. Adatbázis műveletek

Az adatbázis műveletek végrehajtásánál előfordultak olyan esetek, amelyeknél egy adott műveletnek szüksége volt egy másik művelet eredményére. A probléma megoldásában az AsyncTask nevű objektum segített, amellyel háttérszálon lehet futtatni ezeket a műveleteket. A UI-szál azért nem alkalmas erre a feladatra, mert amíg a program ennek a végrehajtásával foglalkozna, a felhasználó nem tudná használni a kezelőfelületet, mert befagyna.

Az AsyncTask doInBackground(Params …) metódusa után meghívódik az onPostExecute(Result) metódus, előbbi háttérszálon, utóbbi már a UI-szálon. Ezt úgy használtam fel, hogy az előbbi metódusban történik az az adatbázis művelet, amelynek az eredményére szüksége van az alkalmazásnak, majd ezt az eredményt paraméterként átadva az utóbbi metódusnak képesek vagyunk egy új AsyncTask-kal felhasználni a kapott eredményt a további műveletekhez.

5. Összefoglalás

A laboratórium témájának az Android-dal való megismerkedés volt az egyik része, rengeteget tanultam annak felépítéséről és felhasználásáról. Az alkalmazást sikerült elkészíteni, amely a meghatározott kezdeti elvárásokat kielégíti. Jelenlegi állapotában használható és helyesen is működik a kijelölt platformon.

További terveim az applikációval kapcsolatban a következők:

- Egyenes kieséses tornáknál amennyiben selejtezőkre kerül sor, az alkalmazás felajánlja a lehetőséget a felhasználónak, hogy kiválassza azokat a csapatokat, amelyeknek majd részt kell venniük a selejtező körben.
- A meccslistáknál érdemes lehet létrehozni egy szűrő felületet, amely tornák, vagy csapatok szerint szűri a meccseket. Ezt például egy automatikus kiegészítéssel rendelkező keresővel lehet megoldani.

- Egy Facebook-os bejelentkező felület létrehozásával lehetősége nyílhat az applikáció felhasználójának, hogy kedvenc csapatainak statisztikáját, vagy bizonyos tornák tabelláját, esetleg egy izgalmas meccs eredményét megossza az idővonalán.
- Adatbázis tekintetében fontos, hogy a bent lévő adatok ne vesszenek el, ha egy új verzió kerül a rendszerbe. Ezért a régi verzióban szereplő adatok migrációja az új verzióba még megoldandó feladat.
- Egy online felület kidolgozása megkönnyítené azon felhasználók dolgát, akik közösen, de külön eszközökön szeretnék használni az applikációt. Ehhez az adatbázis kezelése is egy nagyobb módosítást igényel.
- A felületi elemek fejlesztése, valamint az új elemek felvételénél használt dialógusok felhasználóbarát megjelenítése is a jövőben megvalósítandó feladatok közé tartozik.

6. Külső források

- Android Debug Database: https://github.com/amitshekhariitbhu/Android-Debug-Database
- MPAndroidChart: https://github.com/PhilJay/MPAndroidChart
- MVVM minta: https://medium.com/@husayn.hakeem/android-by-example-mvvm-data-binding-introduction-part-1-6a7a5f388bf7